Université de Montréal

L'algorithme de Branch and Price and Cut pour le problème de conception de réseaux avec coûts fixes et sans capacité

par
Sameh Grainia

Département d'informatique et de recherche opérationnelle
Faculté des arts et des sciences

Mémoire présenté à la Faculté des arts et des sciences
en vue de l'obtention du grade de Maître ès sciences (M.Sc.)
en informatique

Avril, 2015

RÉSUMÉ

Le problème de conception de réseaux est un problème qui a été beaucoup étudié dans le domaine de la recherche opérationnelle pour ses caractéristiques, et ses applications dans des nombreux domaines tels que le transport, les communications, et la logistique.

Nous nous intéressons en particulier dans ce mémoire à résoudre le problème de conception de réseaux avec coûts fixes et sans capacité, en satisfaisant les demandes de tous les produits tout en minimisant la somme des coûts de transport de ces produits et des coûts fixes de conception du réseau. Ce problème se modélise généralement sous la forme d'un programme linéaire en nombres entiers incluant des variables continues. Pour le résoudre, nous avons appliqué la méthode exacte de Branch-and-Bound basée sur une relaxation linéaire du problème avec un critère d'arrêt, tout en exploitant les méthodes de génération de colonnes et de génération de coupes.

Nous avons testé la méthode de Branch-and-Price-and-Cut sur 156 instances divisées en cinq groupes de différentes tailles, et nous l'avons comparée à Cplex, l'un des meilleurs solveurs d'optimisation mathématique, ainsi qu'à la méthode de Branch-and-Cut. Notre méthode est compétitive et plus performante sur les instances de grande taille ayant un grand nombre de produits.

Mots clés: Problème de conception de réseaux, Branch-and-Bound, génération de colonnes, génération de coupes.

ABSTRACT

The network design problem has been studied extensively in the field of operational research given its characteristics and applications in many areas such as transportation, communications, and logistics.

We are particularly interested in solving the multicommodity uncapacitated fixed charge network design problem, with the aim of meeting the demands of all the products while minimizing the total cost of transporting commodities and designing the network. This problem is typically modeled as a linear integer program including continuous variables. To solve it, we applied the exact method of Branch-and-bound based on linear relaxation with a stopping criterion, while exploiting the column generation and cutting-plane methods.

We tested our Branch-and-Price-and-Cut algorithm on 156 instances divided into five groups of different sizes, and we compared it with Cplex, one of the best mathematical optimization solvers. We compare it also with the Branch-and-Cut method.

Numerical results show that our method is competitive and perform better especially on large-scale instances with many commodities.

Keywords: Multicommodity network design problem, Branch-and-Bound, column generation, cutting-plane method.

TABLE DES MATIÈRES

LISTE DES TABLEAUX

LISTE DES FIGURES

DÉDICACE

Que ce travail témoigne de mes respects :

A mon cher père,

À la mémoire de ma mère, et de ma grand-mère,

A mes enfants Aniss et Rami et à toute ma famille.

REMERCIEMENTS

Je tiens tout d'abord à adresser mes remerciements et ma gratitude à mon directeur de recherche, M. Bernard Gendron pour sa patience, son aide et son support tout au long de ce travail.

Je remercie particulièrement Serge Bisaillon pour son énorme aide, son soutien et pour sa grande disponibilité durant mes problèmes d'implémentation de mon programme.

Je remercie également Souhaila El Filali pour toute l'aide qu'elle m'a apporté.

Je remercie sincèrement mon père, mes enfants et toute ma famille, qui malgré l'éloignement, se sont toujours montrées présents, m'ont soutenue et encouragée à toujours suivre ma voie.

Je ne peux m'empêcher en ce moment d'avoir une pensée à ma défunte mère qui avait un grand respect pour les études.

Enfin, que tous ceux et celles que nous avons involontairement oubliés et qui ont participé de près ou de loin à la réalisation de ce modeste travail, trouvent ici l'expression de notre gratitude.

CHAPITRE 1

INTRODUCTION

L'intérêt des problèmes de conception de réseaux réside dans leurs nombreuses applications dans différents domaines, notamment le transport, les communications et la logistique [61]. Ces problèmes se modélisent généralement sous la forme d'un programme linéaire en nombres entiers incluant des variables continues (Mixed Integer Programmming, ou MIP). Ils consistent principalement à construire un réseau (un sous-ensemble d'arcs et de nœuds) pour y transporter un flot (informations, marchandises, ..) à un coût minimum afin de satisfaire une demande.

Dans la pratique, de nombreux problèmes de conception de réseaux rencontrés sont complexes et de grande taille. La difficulté principale dans leur résolution réside dans l'existence des contraintes d'intégralité. Ils sont des problèmes NP-difficiles, raison pour laquelle plusieurs méthodes proposées dans la littérature se sont basées sur les méthodes heuristiques, et seulement les instances de petite taille peuvent être résolues efficacement en un temps raisonnable par des méthodes exactes comme la méthode de Branch-and-Bound.

Dans ce mémoire, nous nous intéressons au problème de conception de réseaux avec coûts fixes et sans capacité (MUND), dont le but est de satisfaire les demandes de tous les produits tout en minimisant la somme des coûts de transport des produits et des coûts fixes de conception. Pour le résoudre, nous avons proposé une méthode exacte de Branch-and-Bound basée sur la relaxation linéaire, tout en exploitant les méthodes de génération de colonnes et de génération de coupes. Nous avons intégré un critère d'arrêt pour suspendre ces méthodes de décomposition avant d'atteindre l'optimalité de la relaxation linéaire si la solution obtenue ne s'améliore pas après un certain nombre d'itérations. Ceci permet d'accélérer la résolution de la relaxation linéaire, ce qui génère en conséquence un gain en temps et permet de résoudre des problèmes de grande taille.

Le chapitre 2 de ce mémoire abordera les définitions de la programmation linéaire, de la programmation linéaire en nombres entiers, et de la relaxation linéaire, et se pour-

suivra en présentant les principales méthodes de résolution de ces problèmes, notamment les méthodes de Branch-and-Bound (B&B), de génération de colonnes et de génération de coupes.

Dans le chapitre 3, nous définissons et décrivons les modèles de conception de réseaux sans et avec capacités, et les différents cas apparentés à ces modèles, puis nous complétons ce chapitre avec une revue de la littérature pertinente au sujet.

Nous décrivons dans le chapitre 4 les différentes étapes de notre algorithme, qui consistent à intégrer les méthodes de génération de colonnes et de génération de coupes avec un critère d'arrêt dans un algorithme de Branch-and-Bound pour résoudre le problème de conception de réseaux avec coûts fixes et sans capacité.

Enfin, nous présentons et interprétons les résultats expérimentaux de notre méthode dans le chapitre 5.

CHAPITRE 2

PROGRAMMATION LINÉAIRE EN NOMBRES ENTIERS

Ce chapitre est consacré à la définition de la programmation linéaire, de la programmation linéaire en nombres entiers, ainsi que la relaxation linéaire. Nous mettons également l'accent sur les principales méthodes de résolution de ces problèmes, notamment les méthodes de Branch-and-Bound (B&B), de génération de colonnes et de génération de coupes.

Ce chapitre est inspiré essentiellement des références suivantes : [2, 6, 23, 26, 44, 49, 68].

2.1 Généralités

Un modèle de programmation mathématique est une description mathématique d'un problème d'optimisation. Un tel problème permet de trouver l'optimum d'une fonction de plusieurs variables (2.1) liées entre elles par des contraintes (2.2), d'où la forme générale suivante [6, 26, 44] :

$$\text{optimiser} \qquad Z = f(x_1, x_2, ..., x_n) \tag{2.1}$$

$$\text{s.à} \qquad h_i(x_1, x_2, ..., x_n) = \left\{ \begin{array}{c} \leq \\ = \\ \geq \end{array} \right\} b_i, \qquad i = 1, 2, ..., m. \tag{2.2}$$

Un modèle de programmation mathématique est classifié selon la nature de sa fonction objectif et de ses contraintes :

- Si f et h_i sont linéaires, on parle d'un modèle de programmation linéaire (PL).
- Si f et h_i sont non linéaires, nous obtenons un modèle de programmation non linéaire (PNL).

- Si f est une fonction quadratique et h_i sont des contraintes linéaires, on définit dans ce cas un modèle de programmation quadratique, un cas particulier des problèmes PNL.

2.2 Programmation linéaire

La programmation linéaire (PL) a été développée en 1947 par G. B. Dantzing et L. Kantorovich, pour répondre à des problèmes complexes que les méthodes classiques n'arrivaient pas à résoudre [1].

La forme générale d'un modèle de programmation linéaire (PL) est donnée comme suit [54] :

$$\begin{array}{ll} \text{optimiser} & c^t x \\ \text{s.à} & Ax \left\{ \begin{array}{c} \leq \\ = \\ \geq \end{array} \right\} b, \\ & x \geq 0. \end{array} \tag{2.3}$$

Avec $c \in R^n, x \in R^n, b \in R^m$: vecteurs,
$A \in R^{m \times n}$: matrice.

2.3 Programmation linéaire en nombres entiers

Dans un modèle de programmation linéaire, si certaines composantes de x sont astreintes à avoir des valeurs entières, on définit alors un modèle de programmation linéaire mixte (MIP) ou « Mixed Integer Program » dont la forme générale est la suivante :

[1] Notons que les méthodes classiques sont des méthodes de résolution basées sur la notion du gradient de la fonction objectif.

$$\begin{array}{ll} \text{optimiser} & c^t x \\ \text{s.à} & Ax \left\{ \begin{array}{c} \leq \\ = \\ \geq \end{array} \right\} b, \\ & x \geq 0, \\ & x \in Z^P \times R^{n-p}. \end{array} \tag{2.4}$$

- Si $p = 0$, (2.4) devient un modèle de programmation linéaire (PL) tel que présenté en (2.3).
- Si $p = n$, (2.4) devient un modèle de programmation linéaire en nombres entiers (IP) ou « Integer Program », et le problème s'écrit comme suit :

$$\begin{array}{ll} \text{optimiser} & c^t x \\ \text{s.à} & Ax \left\{ \begin{array}{c} \leq \\ = \\ \geq \end{array} \right\} b, \\ & x \geq 0, \\ & x \in Z^n. \end{array} \tag{2.5}$$

Dans un modèle de programmation linéaire en nombres entiers, si la variable x est forcée de prendre les valeurs 0 ou 1, on définit alors un modèle de programmation en nombres entiers 0-1, appelé aussi modèle de programmation en nombres binaires (BIP) « Binary Integer Program ».
Ce problème se présente comme suit :

$$\begin{array}{ll} \text{optimiser} & c^t x \\ \text{s.à} & Ax \left\{ \begin{array}{c} \leq \\ = \\ \geq \end{array} \right\} b, \\ & x \geq 0, \\ & x \in B^n. \end{array} \tag{2.6}$$

2.3.1 Relaxation linéaire

La relaxation linéaire (RL), dite aussi « relaxation continue », d'un modèle de programmation linéaire en nombres entiers (IP) est un modèle de programmation linéaire, dont les contraintes d'intégralité sur les variables entières x dans le modèle IP sont relaxées. Elle comporte un ensemble de solutions réalisables plus grand que celui du problème original en nombres entiers.

En générale, le modèle linéaire en nombres entiers (IP) appartient à la classe des problèmes NP difficiles. Cependant, sa relaxation linéaire peut être résolue dans un temps polynomiale, et sa valeur optimale est inférieure ou égale à la valeur optimale du problème IP (dans le cas d'un problème de minimisation). De ce fait, il est intéressant de résoudre la relaxation linéaire pour trouver ou approcher la valeur optimale du problème original (IP).

L'écart relatif entre la valeur optimale du problème IP et la valeur optimale de sa relaxation linéaire s'appelle « écart d'intégralité ».

Soit le modèle IP suivant :

$$(IP): \qquad min = \{c^T x : Ax \geq b, x \geq 0, x \in Z^n\}. \tag{2.7}$$

Sa relaxation linéaire (RL) est définie comme suit :

$$(RL): \qquad min = \{c^T x : Ax \geq b, x \geq 0\}. \tag{2.8}$$

L'écart d'intégralité est calculé par la formule suivante :

$$(Z^* - \tilde{Z})/Z^*. \tag{2.9}$$

où Z^* est la valeur optimale de IP, et $\tilde{Z}$ la valeur optimale de la relaxation linéaire (RL).

2.4 Méthodes de résolution

Les méthodes de résolution les plus efficaces d'un modèle de programmation linéaire en nombres entiers sont basées généralement sur les algorithmes de recherche arborescente par séparation et évaluation (Branch-and-Bound), les méthodes des coupes et les relaxations.
Dans cette section, nous décrivons principalement l'algorithme de Branch-and-Bound. Nous détaillons aussi les méthodes de génération de colonnes et de génération de coupes, qui sont utilisées dans la résolution de la relaxation linéaire.

2.4.1 Algorithme de Branch-and-Bound

Soit le modèle de programmation linéaire en nombres entiers (IP) suivant :

$$\begin{aligned} & Z(P) = min f(x) \\ & x \in S. \end{aligned} \tag{2.10}$$

où $Z(P) > -\infty$, et $S \subseteq R^n$.

L'algorithme de Branch-and-Bound (B&B) consiste à construire nœud par nœud l'arbre d'énumération de l'ensemble des solutions du modèle IP d'une manière intelligente, tout en faisant usage des bornes inférieure et supérieure afin d'éviter la génération de tous les nœuds de l'arbre. Il s'agit principalement de diviser récursivement le problème initial en sous-problèmes plus petits et plus faciles à résoudre : c'est la phase de séparation (*branching*). Quand à la phase d'évaluation (*bound*), elle consiste à identifier les sous-ensembles qui peuvent contenir la solution optimale et à supprimer ceux qui ne la contiennent pas.

L'algorithme de B&B construit et parcourt l'arbre de recherche. Sa racine est le domaine réalisable du problème initial, le modèle IP, tandis que les nœuds représentent les domaines réalisables des sous-problèmes.

À chaque itération, l'algorithme de B&B résout la relaxation du sous-problème courant. Cette dernière fournit une borne inférieure sur la valeur optimale du sous-problème

courant. Différents cas peuvent être rencontrés :

- Le sous-problème n'est pas réalisable, cela signifie qu'il n'y a pas de solution dans cette branche. Elle est exclue.
- La borne inférieure du nœud est supérieure à la meilleure solution réalisable obtenue par l'algorithme de B&B : on peut alors affirmer que la solution optimale globale ne peut pas être contenue dans le sous ensemble de solutions représenté par ce nœud. On l'élague.
- La solution de la relaxation continue du nœud devient la meilleure solution réalisable du problème initial si elle est entière, et que sa valeur optimale est inférieure à celle obtenue par l'algorithme de B&B jusqu'à présent.
- Finalement, si la borne inférieure obtenue est meilleure que la dernière valeur obtenue par l'algorithme de B&B, mais sa solution n'est pas entière, le problème sera divisé en sous-problèmes.

La méthode sera appliquée récursivement au reste des nœuds, et elle s'arrête lorsqu'il n'y a plus de nœud à évaluer.
La méthode de B&B est résumée dans la Figure 2.1.

Entrée : problème de minimisation P.
Sortie : x^* solution optimale de P, Z^* valeur optimale.

1. Initialisation :
   $Z^* = \infty$, $\Gamma = \{P\}$.
2. Répéter jusqu'à $\Gamma = \emptyset$.
3. Sélectionner un nœud $R \in \Gamma$, $\Gamma \leftarrow \Gamma \backslash \{R\}$.
4. Évaluer R :
   Résoudre la relaxation R_{RL} de R.

- Si R_{RL} est non réalisable, supprimer R de Γ et aller à l'étape 2.
- Sinon soit : x^R la solution optimale de R_{RL}, et Z^R_{RL} sa valeur optimale.

5. Élaguer :
   - Si $Z^R_{RL} \geq Z^*$, supprimer R de Γ et retourner à l'étape 2.
   - Sinon
     - Si x^R n'est pas entière, aller à l'étape 6.
     - Sinon $Z^* := Z^R_{RL}$, $x^* := x^R$. Aller à l'étape 2.
6. Brancher :
   Séparer R en sous-problèmes : $R = R_1 \bigcup R_2 \bigcup ... \bigcup R_k$.
   $\Gamma := \Gamma \bigcup \{R_1, R_2, .., R_k\}$. Aller à l'étape 2.

Figure 2.1 – Algorithme de Branch-and-Bound.

Pendant ce processus, à chaque nœud, l'algorithme doit avoir une borne inférieure calculée par une relaxation, et une borne supérieure identifiée par une solution réalisable. Si ces deux bornes sont égales, une solution optimale est trouvée, et le processus s'arrête. Les bornes obtenues sont utilisées pour orienter et accélérer la phase d'exploration de l'algorithme de Branch-and-Bound. Des bonnes techniques de calcul des bornes permettent alors d'élaguer le plus tôt possible des branches qui ne sont pas intéressantes.

L'évaluation d'un nœud, la sélection du prochain nœud à évaluer et la sélection de la variable de branchement sont les principaux composants qui influencent fortement la performance de l'algorithme de Branch-and-Bound.

Dans les sections suivantes, nous parlerons de ces composants séparément.

2.4.1.1 L'évaluation d'un nœud

L'évaluation d'un nœud d'un arbre de recherche a pour but de déterminer l'optimum de l'ensemble des solutions réalisables associé au nœud en question. Elle permet aussi de

réduire l'espace de recherche en éliminant les nœuds qui ne contiennent pas la solution optimale. Pour ce faire, l'algorithme se dote d'une fonction pour mettre une borne sur certaines solutions afin de les exclure ou de les maintenir comme des solutions potentielles. De ce fait, deux techniques différentes de relaxation peuvent être utilisées dans l'algorithme de B&B :
- La relaxation lagrangienne, qui consiste à relâcher certaines contraintes compliquantes (difficiles) et de les réintroduire dans la fonction objectif, pondérées par des coefficients que l'on appelle multiplicateurs de Lagrange. Notons toute fois, qu'il faut adapter la condition d'arrêt sur l'intégralité de la solution, et de la remplacer par la satisfaction des contraintes relaxées et des conditions d'écart complémentaires.
- La relaxation linéaire qui consiste à relaxer les contraintes d'intégralité sur les variables entières. Le modèle linéaire ainsi obtenu est généralement résolu par la méthode dual du simplexe, étant donné que la solution optimale du dual de la relaxation linéaire du nœud parent reste réalisable pour les duaux des relaxations linéaires des nœuds fils.

Pour réduire le temps d'évaluation d'un nœud et améliorer son efficacité, des méthodes de décomposition comme la génération de colonnes et la génération de coupes sont utilisées. Toutes les méthodes de décomposition se basent sur le même principe : le modèle original est décomposé en plusieurs sous-problèmes plus faciles, généralement coordonnés par un problème maître.
Nous présentons dans ce qui suit deux techniques qui utilisent ce principe.

Méthode de génération de colonnes : La génération de col−onnes est l'une des principales méthodes utilisées pour résoudre des modèles de programmation linéaire ayant un grand nombre de variables. Elle décompose le modèle en plusieurs sous-problèmes plus petits, coordonnés par le problème maître. Son principe consiste à ne pas considérer explicitement toutes les variables du problème maître, mais seulement un ensemble réduit de variables, ce qui donne lieu à ce qu'on appelle le problème maître restreint (PMR).

Considérons un sous-ensemble de variables x du modèle (2.8) pour définir le problème maître restreint, et soit $I' \subseteq I$ l'ensemble des indices des variables x incluses dans

le problème PMR.

(PMR)

$$\begin{aligned} \min \quad & c^t x \\ \text{s.à} \quad & Ax \geq b, \\ & x_i \geq 0, i \in I' \\ & x_i = 0, i \in I \backslash I'. \end{aligned} \tag{2.11}$$

À chaque itération, le problème maître restreint est résolu à l'optimum. La méthode de génération de colonnes doit vérifier si la valeur optimale du PMR l'est aussi pour le problème maître (PM). Pour ainsi faire, la méthode vérifie s'il existe au moins une variable x_i, telle que $i \in I \backslash I'$ à ajouter au problème PMR qui pourrait améliorer la solution du modèle PM. La résolution du sous-problème définie par le dual du problème maître restreint a pour but de déterminer et d'ajouter de nouvelles variables au problème PMR en calculant leurs coûts réduits. Si au moins une variable x_i, pour $i \in I \backslash I'$ a un coût réduit négatif, cela signifie que cette variable doit être ajoutée au problème PMR, et ce dernier est résolu à nouveau. Dans le cas contraire, la solution optimale du problème maître restreint est aussi optimale pour le modèle (2.8), et la génération de colonnes s'arrête.

L'algorithme de la génération de colonnes est présenté dans la Figure 2.2.

1. Choisir un sous ensemble de variables x_i, $i \in I' \subseteq I$.
2. Résoudre le problème maître restreint correspondant.
3. Résoudre le sous-problème pour obtenir les coûts réduits des variables $x_i, i \in I \backslash I'$.
4. S'il existe au moins une variable qui a un coût réduit négatif, alors l'introduire dans le problème maître restreint et aller à l'étape 2.
5. Sinon arrêter.

Figure 2.2 – Méthode de génération de colonnes.

La méthode de génération de colonnes combinée à la méthode de Branch-and-Bound, qu'on appelle Branch-and-Price, est appliquée pour résoudre les modèles linéaires en nombres entiers ayant un très grand nombre de variables. À chaque nœud de l'arbre de recherche, la méthode de génération de colonnes fait appel itérativement à un problème maître restreint qui contient un ensemble réduit de variables, et à des sous-problèmes qui permettent de déterminer les variables à ajouter au problème PMR.

Méthode de génération de coupes : La difficulté de la résolution d'un modèle linéaire en nombres entiers réside principalement dans l'identification des contraintes linéaires qui définissent l'enveloppe convexe du domaine réalisable, notamment pour des problèmes de grande taille, sachant que la solution optimale du modèle est un point extrême de l'enveloppe convexe. Ces contraintes sont appelées « inégalités valides fortes ».

Rappelons qu'une inégalité valide est une contrainte qui est vérifiée par n'importe quelle solution réalisable du modèle linéaire en nombres entiers. Si, en plus, elle élimine des solutions du domaine de la relaxation linéaire, on dit qu'elle est une coupe.

L'introduction des inégalités valides a pour but d'améliorer la formulation et d'obtenir de meilleures bornes inférieures sur la valeur optimale du modèle linéaire en nombres entiers. Souvent l'ajout de toutes les inégalités valides au modèle n'est pas efficace, étant donné qu'il devient difficile de résoudre le modèle résultant dans un temps raisonnable. L'idée de la méthode de génération de coupes est de résoudre une relaxation du modèle, et si la solution optimale n'est pas réalisable pour le problème original, à générer et à ajouter itérativement des inégalités valides violées par la solution courante au modèle relaxé, en éliminant uniquement les solutions fractionnaires et aucune solution entière, et ce jusqu'à ce qu'il n'y en ait plus d'inégalités valides violées.

En effet, l'ajout des inégalités valides au problème relaxé réduit son domaine des solutions réalisables sans toutefois modifier celui du problème linéaire en nombres entiers, ce qui améliore sa borne inférieure, et bien évidement réduit l'écart d'intégralité qui influe positivement sur la performance de l'algorithme de Branch-and-Bound.

Les inégalités valides sont identifiées par la résolution du problème de séparation, qui permet de déterminer si la solution courante appartient à l'enveloppe convexe des

solutions entières réalisables. Dans le cas contraire, il génère des coupes séparant cette solution de l'enveloppe convexe.

La méthode de génération de coupes est présentée dans la Figure 2.3.

1. Soit $\tilde{P}$ la relaxation du problème linéaire en nombres entiers (P).
2. Résoudre $\tilde{P}$.
   Soit $\tilde{x}$ sa solution optimale.
3. Si $\tilde{x}$ est entière, aller à l'étape 6.
4. Résoudre le problème de séparation, et identifier les inégalités valides violées.
   S'il n'y en a pas, aller à l'étape 6.
5. Ajouter les inégalités valides au problème $\tilde{P}$. Aller à l'étape 1.
6. Fin.

Figure 2.3 – Méthode de génération de coupes.

La méthode de génération de coupes, combinée à la méthode de Branch-and-Bound, qu'on appelle Branch-and-Cut est appliquée pour réduire le domaine réalisable de la relaxation linéaire, et améliorer la borne inférieure du modèle linéaire en nombres entiers. À chaque nœud de l'arbre de recherche, la méthode de génération de coupes résout la relaxation, et ajoute itérativement de nouvelles inégalités valides violées par la solution non entière de la relaxation pour améliorer la solution courante.

Algorithme de Branch-and-Price-and-Cut : Lorsque la génération de colonnes et la génération de coupes sont combinées et intégrées dans le processus de Branch-and-Bound, elles donnent lieu à la méthode de Branch-and-Price-and-Cut. À chaque nœud de l'arbre de recherche, les méthode de génération de colonnes et de génération de coupes,

qu'on appelle Price-and-Cut, sont appliquées pour résoudre la relaxation linéaire, de sorte qu'on résout alternativement le sous-problème d'évaluation, qui permet de générer des colonnes améliorantes, et le sous-problème de séparation, qui génère des inégalités valides violées par la solution fractionnaire.

2.4.1.2 Règles de branchement

Le problème de sélection des variables sur lesquelles brancher revient à décider comment partitionner un nœud parent pour construire l'arbre de recherche. Autrement dit, déterminer les variables sur la base desquelles le nœud parent doit se ramifier afin de pouvoir créer les enfants.

La règle de branchement divise le domaine réalisable du nœud parent R en sous-problèmes R_i en éliminant les solutions fractionnaires, et en garantissant de garder toutes les solutions entières du domaine réalisable.

Pour les problèmes de programmation linéaires en nombres entiers, les nœuds enfants se construisent en ajoutant séparément les contraintes de types $x_i \geq \lceil x_i \rceil$ et $x_i \leq \lfloor x_i \rfloor$, pour une variable fractionnaire sélectionnée x_i dans le problème linéaire du nœud parent.

Le choix d'une bonne règle de branchement peut améliorer de façon spectaculaire l'efficacité de l'algorithme de Branch-and-Bound, de telle façon qu'elle génère moins de nœuds dans l'arbre de recherche, et améliore le plus rapidement possible les bornes inférieures du problème.

On présente par la suite certaines règles de branchement les plus utilisées par les meilleurs logiciels d'optimisation, et qui s'adressent particulièrement aux modèles de programmation linéaire en variables binaires, et qui basent plus principalement sur la relaxation linéaire à chaque nœud de l'arbre de recherche. Notons que les contraintes à introduire pour construire les nœuds enfants dans un problème BIP seront de types : $x_i = 0$ et $x_i = 1$.

Variable la plus fractionnaire : C'est le choix le plus classique, cette règle sélectionne la variable ayant la partie fractionnaire la plus proche de 0,5, c'est-à-dire, choisir x_i telle que $i = \text{argmin}_{i \in N}\{x_i - \lfloor x_i \rfloor, \lceil x_i \rceil - x_i\}$.

Selon [2], la performance de cette règle n'est généralement pas mieux que de choisir la

variable de manière aléatoire.

Variable la moins fractionnaire : Contrairement à la variable la plus fractionnaire, cette règle choisit la variable qui est proche de l'intégralité, ce qui correspond cette fois-ci à choisir x_i, telle que $i = \text{argmax}_{i \in N}\{x_i - \lfloor x_i \rfloor, \lceil x_i \rceil - x_i\}$. Cette règle, comme la règle de la variable la plus fractionnaire, a une très mauvaise performance [2].

Le branchement fort : Le principe de cette règle est de résoudre les deux sous problèmes relaxés produits après une fixation de la variable fractionnaire à 1 et à 0 respectivement, et de sélectionner le nœud qui a le meilleur accroissement de la borne inférieure. En revanche, ce n'est généralement pas pratique de l'appliquer en raison du grand nombre de calculs qu'elle requiert. Son effort de calcul peut toutefois être limité de deux manières. D'une part, on peut définir un ensemble plus petit de candidats sur lesquels il faut brancher. D'autre part, on ne résout les relaxations linéaires que partiellement, en effectuant seulement un nombre limité d'itérations de la méthode duale du simplexe.

Pseudo-coûts : Cette règle conserve l'historique des branchements déjà effectués sur chaque variable comme une indication de la qualité de la variable elle-même. En effet, elle calcule la moyenne des accroissements de la borne inférieure lorsqu'on fixe cette variable à une valeur entière (0 ou 1) tout au long de son branchement.

Étant donné σ_i^0 et σ_i^1, qui correspondent aux accroissements unitaires de la borne inférieure lorsqu'on branche sur une variable fixée à 0 et à 1 respectivement, et qui sont définis comme suit :

$$\sigma_i^0 = \frac{\Delta_i^0}{x_i - \lfloor x_i \rfloor} \qquad \text{et} \qquad \sigma_i^1 = \frac{\Delta_i^1}{\lceil x_i \rceil - x_i}$$

Sachant que Δ_i^0 et Δ_i^1 définissent l'accroissement de la borne inférieure, les pseudo-coûts ρ_i^0 et ρ_i^1 sont calculés comme suit :

$$\rho_i^0 = \frac{\varsigma_i^0}{\eta_i^0} \qquad \text{et} \qquad \rho_i^1 = \frac{\varsigma_i^1}{\eta_i^1}$$

où ς_i^0 et ς_i^1 représentent la somme des accroissements unitaires de la borne lorsqu'on branche sur la variable x_i fixée à 0 et à 1, et η_i^0 et η_i^1 représentent le nombre de branchements effectués sur la variable x_i en la fixant à l'une ou l'autre de ces valeurs.

Une fois les pseudo-coûts déterminés, la variable sur laquelle brancher est la variable x_i telle que : $i = \text{argmax}_{i \in N}\{\lambda . \min\{\rho_i^0, \rho_i^0\} + (1-\lambda). \max\{\rho_i^0, \rho_i^1\}\}$

où λ est un paramètre dans l'intervalle $[0,1]$, généralement choisi proche de 1.
Il est à noter qu'il existe d'autres façons pour calculer les pseudo-coûts ainsi que pour sélectionner la variable sur laquelle brancher (le lecteur est invité à consulter l'article [59]).

Nous pouvons observer qu'au début de l'algorithme, $\varsigma_i^0 = \eta^0 = \varsigma_i^1 = \eta^1 = 0, \forall i \in N$. Si le branchement à 0 ou à 1 n'est pas défini i.e., $\eta_i^0 = 0$ où $\eta_i^1 = 0$, les pseudo-coûts ρ_i^0 et ρ_i^1 sont égaux en ce moment à la moyenne des pseudo coûts des autres variables initialisées. Si aucun pseudo-coût n'est défini, cette moyenne est fixée à 1. De plus, cette règle n'utilise que peu d'information locale pour prendre une décision. Malgré ces inconvénients, cette règle de branchement est rapide et reste meilleure que la règle de la variable la plus fractionnaire [2].

Pseudo-coûts avec initialisation par branchement fort : Comme mentionné auparavant, la faiblesse de la méthode des pseudo-coûts est qu'au début, aucune information n'est disponible pour calculer le pseudo-coût tant qu'on n'a pas branché sur aucune variable. Tandis que pour la règle de branchement fort, le temps de calcul reste toujours élevé, même avec les accélérations intégrées. L'idée de la règle des pseudo-coûts avec initialisation par branchement fort est d'appliquer la règle de branchement fort sur les variables dont les pseudo-coûts n'ont pas encore été définis afin de les initialiser, et d'utiliser ensuite la règle des pseudo-coûts pour accélérer les calculs.

Branchement fiable : Le branchement fiable est une généralisation de la règle précédente, où le branchement fort est appliqué non seulement sur les variables non initialisées, mais aussi sur des variables ayant des valeurs de pseudo-coûts dits « non fiables ».

L'idée est de définir un seuil de fiabilité, c'est-à-dire un niveau en dessous duquel les informations des pseudo-coûts ne sont pas considérées comme suffisamment précises. Un tel seuil est principalement associé avec le nombre de décisions de branchement précédentes impliquant la variable.

2.4.1.3 Règles d'exploration de l'arbre

Le nombre de nœuds explorés dans un algorithme de Branch-and-Bound détermine de façon importante le temps de résolution d'un problème. À chaque itération, l'algorithme de Branch-and-Bound doit sélectionner un nœud à explorer parmi les nœuds qui n'ont pas encore été évalués. Pour ce faire, il existe deux méthodes de parcours essentielles : la méthode meilleur d'abord, et la méthode en profondeur.

La méthode meilleur d'abord : Cette méthode consiste à sélectionner le nœud ayant la plus petite borne inférieure parmi les nœuds non évalués tout en améliorant la borne inférieure du problème. Lors de la création des nœuds, les bornes de la relaxation linéaire sont évaluées immédiatement, raison pour laquelle on l'appelle une évaluation hâtive. De plus, cette stratégie garantit d'explorer seulement les branches de l'arbre de recherche dont la borne inférieure est plus petite que la valeur optimale, ce qui réduit au minimum le nombre de nœuds générés dans l'arbre de recherche. Par contre, la méthode utilise plus d'espace mémoire parce qu'elle élargit l'arbre dans toutes les directions en gardant un très grand nombre de nœuds actifs. De plus, elle prend plus de temps pour la résolution des sous-problèmes vu que chacun d'entre eux n'est pas sélectionné de façon hiérarchique dans l'arbre de recherche, ce qui nécessite de les résoudre à nouveau.

La méthode en profondeur : La méthode en profondeur favorise la recherche dans les nœuds les plus éloignés de la racine, c'est-à-dire qu'elle sélectionne le nœud le plus profond dans l'arbre de recherche. Cette règle choisit le nœud le plus récemment généré. Il s'agit d'un nœud enfant si le nœud courant est branché, ou d'un nœud du même niveau ou du niveau supérieur si le nœud courant est élagué, ou s'il a fourni une meilleure solution réalisable. Cette technique s'appelle le retour arrière (*backtracking*).

Lors de la création des nœuds, les bornes ne sont évaluées que lorsque le nœud est sélectionné, on l'appelle de ce fait une évaluation paresseuse. Contrairement à la méthode meilleur d'abord, la méthode en profondeur mène rapidement à une solution optimale en réduisant le temps de calcul et les besoins en mémoire. Elle aide aussi la réoptimisation en améliorant la borne supérieure. Son inconvénient est qu'elle génère un grand nombre de nœuds, notamment si les bornes supérieures ne sont pas assez bonnes.

Recherche hybride : La recherche hybride est une combinaison de la recherche meilleur d'abord et de la recherche en profondeur. Elle consiste à choisir les nœuds ayant la meilleure borne inférieure (recherche meilleur d'abord) en premier ce qui permet d'améliorer la borne inférieure, puis d'appliquer la recherche en profondeur sur l'un des nœuds enfants afin de trouver rapidement des solutions réalisables.

CHAPITRE 3

CONCEPTION DE RÉSEAUX

Dans ce chapitre, nous étudions les problèmes de conception de réseaux sans et avec capacités. Pour chacun de ces problèmes, nous présentons plusieurs modèles, différents cas apparentés à ces modèles, et nous concluons par une revue de la littérature pertinente.

3.1 Conception de réseaux avec coûts fixes et sans capacité (MUND)

3.1.1 Présentation du problème et notation

Soit $G = (V, A)$ un graphe orienté où V est l'ensemble des sommets, A l'ensemble des arcs et K l'ensemble des produits qui circulent dans G. Chaque arc est représenté par une paire (i, j) où i et j sont ses extrémités. Pour chaque produit k, $k = 1...K$, nous distinguons $O(k)$ le sommet d'origine, $D(k)$ le sommet de destination, et $d^k > 0$ la quantité du produit k à transporter de $O(k)$ à $D(k)$. Deux types de coûts sont associés à ce graphe : $c_{ij}^k \geq 0$ est le coût unitaire de transport du produit k sur l'arc (i, j), et $f_{ij} \geq 0$ est un coût imposé si au moins un produit traverse l'arc (i, j).

Le problème consiste à satisfaire les demandes de tous les produits tout en minimisant la somme des coûts de transport de produits et des coûts fixes d'ouverture des arcs [16, 26].

3.1.2 Modèles

3.1.2.1 Formulations générales

Le problème de conception de réseaux avec coûts fixes et sans capacité (MUND) ou « Multicommidity Uncapacitated Network Design » se modélise généralement sous la forme d'un programme linéaire en nombres entiers incluant des variables continues (MIP).

Deux types de variables sont introduits :

- $x_{ij}^k \geq 0$, des variables continues représentant la quantité de flot du produit k qui circule sur l'arc (i, j) ;

- y_{ij}, des variables binaires représentant la décision d'ouverture de l'arc (i, j) :

$$y_{ij} = \begin{cases} 1 & \text{, si l'arc } (i,j) \text{ est ouvert,} \\ 0 & \text{, sinon.} \end{cases}$$

On dénote par V_i^+ et V_i^- les ensembles :
$V_i^+ = \{j \in V | (i,j) \in A\}$ et $V_i^- = \{j \in V | (j,i) \in A\}$.

Le modèle de programme mathématique peut s'écrire de la façon suivante :

$$\min \quad \sum_{k \in K} \sum_{(i,j) \in A} c_{ij}^k x_{ij}^k + \sum_{(i,j) \in A} f_{ij} y_{ij} \tag{3.1}$$

Sujet à

$$\sum_{j \in V_i^+} x_{ij}^k - \sum_{j \in V_i^-} x_{ji}^k = \begin{cases} d^k, & \text{si } i = O(k), \\ -d^k, & \text{si } i = D(k), \\ 0, & \text{sinon.} \end{cases} \qquad \forall i \in V, k \in K, \tag{3.2}$$

$$\sum_{k \in K} x_{ij}^k \leq (\sum_{k \in K} d^k) y_{ij}, \qquad \forall (i,j) \in A, \tag{3.3}$$

$$x_{ij}^k \geq 0, \qquad \forall (i,j) \in A, k \in K, \tag{3.4}$$

$$y_{ij} \in \{0,1\}, \qquad \forall (i,j) \in A. \tag{3.5}$$

Les contraintes (3.2) sont les contraintes de conservation de flot qui imposent que les quantités de flot reçues et émises par chaque sommet de transfert soient égales, sauf pour les sommets d'origine et de destination. Les contraintes (3.3) garantissent qu'aucun flot ne peut circuler sur un arc qui n'est pas ouvert. Lorsqu'un arc (i, j) est ouvert, la quantité de flot de tous les produits qui circulent sur l'arc ne dépasse pas la somme des demandes de tous les produits, $\sum_{k \in K} d^k$. Les contraintes (3.4) sont les contraintes de non-négativité

sur les variables x_{ij}^k, et les contraintes (3.5) sont les contraintes d'intégralité sur les variables y_{ij}, et qui leurs imposent de prendre des valeurs binaires.

Étant donné que $x_{ij}^k \leq d^k$, $\forall (i,j) \in A, k \in K$, i.e., la quantité du produit k qui circule sur l'arc (i,j) ne dépasse pas la quantité demandée d^k, les contraintes (3.3) peuvent être désagrégées en $|K|$ contraintes de la forme :

$$x_{ij}^k \leq d^k y_{ij}, \qquad \forall (i,j) \in A, k \in K. \tag{3.6}$$

Ces contraintes tout comme (3.3), assurent que le flot de tout produit soit nul sur l'arc (i,j) dans le cas où ce dernier n'est pas ouvert ($y_{ij} = 0$). Cependant, ces contraintes permettent d'améliorer la qualité des bornes inférieures de la relaxation linéaire. La formulation obtenue en éliminant les contraintes (3.3) et en les remplaçant par les contraintes (3.6) est appelée la formulation forte. Elle se présente sous la forme suivante :

$$\min \quad \sum_{k \in K} \sum_{(i,j) \in A} c_{ij}^k x_{ij}^k + \sum_{(i,j) \in A} f_{ij} y_{ij} \tag{3.7}$$

Sujet à

$$\sum_{j \in V_i^+} x_{ij}^k - \sum_{j \in V_i^-} x_{ji}^k = \begin{cases} d^k, & \text{si } i = O(k), \\ -d^k, & \text{si } i = D(k), \\ 0, & \text{sinon.} \end{cases} \qquad \forall i \in V, k \in K, \tag{3.8}$$

$$x_{ij}^k \leq d^k y_{ij}, \qquad \forall (i,j) \in A, k \in K, \tag{3.9}$$

$$x_{ij}^k \geq 0, \qquad \forall (i,j) \in A, k \in K, \tag{3.10}$$

$$y_{ij} \in \{0,1\}, \qquad \forall (i,j) \in A. \tag{3.11}$$

À l'inverse, la formulation (3.1) à (3.5) présentée précédemment, où les contraintes (3.3) sont conservées, s'appelle la formulation faible.

3.1.2.2 Formulations équivalentes

Dans les modèles suivants, $C_{ij}^k \geq 0$ représente le coût total de transport du produit k sur l'arc (i, j), qui est égal à $c_{ij}^k d^k$, où c_{ij}^k est le coût unitaire de transport du produit k sur l'arc (i, j) comme défini dans la section précédente. La variable $x_{ij}^k \geq 0$ représente maintenant la fraction de la demande du produit k qui circule sur l'arc (i, j) [33, 46].

Nous présentons les formulations faible et forte associées à cette redéfinition des variables de flot.

- Formulation faible :

$$\min \quad \sum_{k\in K}\sum_{(i,j)\in A} C_{ij}^k x_{ij}^k + \sum_{(i,j)\in A} f_{ij} y_{ij} \tag{3.12}$$

Sujet à

$$\sum_{j\in V_i^+} x_{ij}^k - \sum_{j\in V_i^-} x_{ji}^k = \begin{cases} 1, & \text{si } i = O(k), \\ -1, & \text{si } i = D(k), \\ 0, & \text{sinon.} \end{cases} \qquad \forall i \in V, k \in K, \tag{3.13}$$

$$\sum_{k\in K} x_{ij}^k \leq |K| y_{ij}, \qquad \forall (i, j) \in A, \tag{3.14}$$

$$x_{ij}^k \geq 0, \qquad \forall (i, j) \in A, k \in K, \tag{3.15}$$

$$y_{ij} \in \{0, 1\}, \qquad \forall (i, j) \in A. \tag{3.16}$$

- Formulation forte :

$$\min \quad \sum_{k\in K}\sum_{(i,j)\in A} C_{ij}^k x_{ij}^k + \sum_{(i,j)\in A} f_{ij} y_{ij} \tag{3.17}$$

Sujet à

$$\sum_{j\in V_i^+} x_{ij}^k - \sum_{j\in V_i^-} x_{ji}^k = \begin{cases} 1, & \text{si } i = O(k), \\ -1, & \text{si } i = D(k), \\ 0, & \text{sinon.} \end{cases} \qquad \forall i \in V, k \in K, \tag{3.18}$$

$$x_{ij}^k \leq y_{ij}, \qquad \forall (i,j) \in A, k \in K, \tag{3.19}$$

$$x_{ij}^k \geq 0, \qquad \forall (i,j) \in A, k \in K, \tag{3.20}$$

$$y_{ij} \in \{0,1\}, \qquad \forall (i,j) \in A. \tag{3.21}$$

3.1.2.3 Formulation Chemin-Produit

Les formulations présentées dans la section 3.1.2.2 sont appelées formulations Arc-Produit, car le flot est représenté en fonction des arcs et des produits. En revanche, la formulation Chemin-Produit est basée principalement sur l'ensemble P^k, $k \in K$ de tous les chemins existants entre $O(k)$ et $D(k)$ dans le réseau G. Désignons par x_p^k la variable de décision correspondant à la fraction de la demande du produit k à faire passer par le chemin p de P^k, qui est égale à $\sum_{(i,j)\in A} \delta_{ij}^{kp} x_{ij}^k$, sachant que δ_{ij}^{kp} est un paramètre qui vaut 1 si l'arc (i,j) fait partie du chemin p, et 0 sinon. Le coût de transport est associé dans ce cas à chaque chemin $p \in P^k$, et il est calculé par la formule : $C_p^k = \sum_{(i,j)\in A} \delta_{ij}^{kp} C_{ij}^k$.

La formulation forte Chemin-Produit prend la forme suivante (nous ....ttons la formulation faible,qui est facile à ....) :

$$\min \quad \sum_{k\in K} \sum_{p\in P^k} C_p^k x_p^k + \sum_{(i,j)\in A} f_{ij} y_{ij} \tag{3.22}$$

Sujet à

$$\sum_{p\in P^k} x_p^k = 1, \qquad \forall k \in K, \tag{3.23}$$

$$\sum_{p \in P^k} \delta_{ij}^{kp} x_p^k \leq y_{ij}, \qquad \forall (i,j) \in A, k \in K, \tag{3.24}$$

$$x_p^k \geq 0, \qquad \forall p \in P^k, k \in K, \tag{3.25}$$

$$y_{ij} \in \{0,1\}, \qquad \forall (i,j) \in A. \tag{3.26}$$

Dans ce modèle, nous retrouvons le même type de contraintes que dans le modèle Arc-Produit. Les contraintes (3.23) représentent les contraintes de conservation de flot, les contraintes (3.24) imposent un flot nul si l'arc n'est pas ouvert, et les contraintes (3.25) et (3.26) sont les contraintes de non-négativité et d'intégralité respectivement.

Malgré que le nombre de contraintes de conservation de flot de la formulation Arc-Produit (qui est égal à $|V||K|$) est plus grand que celui de la formulation Chemin-Produit (qui est égal à $|K|$), le nombre de chemins qui relient les paires origine-destination est exponentiel, ce qui nécessite généralement d'appliquer la méthode de génération de colonnes pour résoudre cette formulation.

3.1.3 Cas particuliers

Plusieurs problèmes apparentés au problème présenté dans la section 3.1 sont considérés comme des cas particuliers. Nous présentons dans cette section le problème de localisation sans capacité et le problème du plus court chemin. Pour d'autres cas particuliers, nous référons le lecteur à la référence [61].

Le problème de localisation sans capacité : Le problème consiste à installer un certain nombre de dépôts parmi un sous-ensemble de sites dans le but de satisfaire la demande de clients, tout en minimisant les coûts totaux de transport des produits et les coûts fixes d'installation des dépôts. Ce problème peut être formulé comme un problème de conception de réseaux. On définit le graphe $G = (V,A)$, où V est l'ensemble des nœuds

qui représentent les sites possibles pour l'installation des dépôts ainsi que les clients à servir, et A est l'ensemble des arcs. Nous introduisons une super-origine S, de telle sorte qu'elle fournisse toutes les demandes des produits au réseau et qu'elle soit reliée à chaque dépôt par un arc. À chacun de ces arcs, on associe un coût fixe de conception égal à f_j, et un coût de transport de produit nul ($c^k_{Sj} = 0$). La résolution de ce problème revient à résoudre le problème de localisation sans capacité.

Le problème du plus court chemin : Ce problème consiste à relier un sommet source S à tous les autres sommets i par le plus court chemin (de S vers i), où la longueur d'un chemin est la somme des longueurs c_{ij} de chaque arc (i, j) qui compose le chemin. Le problème du plus court chemin peut être vu comme un problème de conception de réseaux sans capacité. En effet, dans un graphe $G = (V, A)$, nous considérons l'ensemble des produits $K = i$, $i \in V \backslash \{S\}$ ayant la même source $O(k) = S$, $\forall k \in K$, des différentes destinations $D(k)$, $\forall k \in K$, et une demande $d_k = 1$. Chaque arc a un coût d'installation nul $f_{ij} = 0$, $\forall (i, j) \in A$. La solution optimale de ce problème correspond à la solution optimale du problème du plus court chemin d'un sommet vers tous les autres sommets.

3.1.4 Revue de littérature pour le problème de conception de réseaux avec coûts fixes et sans capacité

Le problème de conception de réseaux avec coûts fixes et sans capacité (MUND) appartient à la classe des problèmes NP-difficiles comme mentionné par Magnanti et Wong [61], et par Balakrishnan et al. [6]. Krarup et Pruzan [54] ont démontré que le problème de localisation sans capacité, qui représente un cas particulier du MUND, est un problème NP-difficile.

Ce problème a poussé les chercheurs à élaborer des méthodes relativement efficaces pour le résoudre, des méthodes exactes qui garantissent l'optimalité dans un intervalle de temps bien déterminé mais exponentiel, ainsi que des méthodes heuristiques qui consistent à trouver une solution proche de l'optimum en un temps raisonnable.

3.1.4.1 Méthodes exactes

Tout comme le cas des modèles de programmation linéaire en nombres entiers, l'algorithme de Branch-and-Bound, la décomposition de Benders et la relaxation lagrangienne sont les méthodes exactes appliquées pour résoudre le problème MUND.

Balakrishnan et al. [6] ont appliqué la méthode duale-ascendante pour le problème de conception de réseau sans capacité. Elle est considérée parmi les méthodes de résolution les plus efficaces (un gap de 1% à 4% de l'optimum sur des problèmes de grande taille). Cette méthode est une généralisation de l'algorithme de Dijkstra [24] pour le problème du plus court chemin, et de l'algorithme d'Erlenkotter [28] pour le problème de localisation sans capacité. Elle vise à générer de bonnes bornes inférieures de manière relativement rapide en résolvant le dual de la relaxation linéaire, suivie d'une procédure d'ajustement afin d'obtenir une solution duale améliorée. La méthode utilise aussi une heuristique de recherche locale pour trouver des solutions primales réalisables. Ces dernières peuvent aider à réduire la taille du problème en éliminant certaines variables de conception.

Holmberg et Hellstrand [46] ont proposé une méthode exacte de Branch-and-Bound basée sur une heuristique lagrangienne. Son principe consiste à résoudre la relaxation lagrangienne du problème appliquée aux contraintes de conservation de flot, en résolvant le dual lagrangien avec la méthode du sous-gradient combinée avec une heuristique primale pour trouver des solutions réalisables du problème. Les résultats expérimentaux ont démontré l'efficacité de la méthode à résoudre des problèmes de grande taille.

Magnanti et al. [62] ont montré que plusieurs inégalités valides présentées dans la littérature pour améliorer la borne inférieure peuvent être définies comme des coupes de Benders. De plus, ils ont résolu le problème avec une version accélérée de la décomposition de Benders [8] combinée à la méthode duale-ascendante, où ils ont dérivé des nouvelles coupes de Benders.

Plusieurs chercheurs ont appliqué et amélioré l'algorithme de Branch-and-Bound sur des problèmes sans capacité de moyenne taille, avec et sans contrainte de budget. Cette dernière contrainte peut être formulée comme suit :

$$\sum_{(i,j)\in A} e_{ij} y_{ij} \leq B, \tag{3.27}$$

où e_{ij} représente la quantité d'unités budgétaires consommées par l'ouverture de l'arc (i, j).

Boyce et al. [12] ont proposé un algorithme de Branch-and-Bound après avoir dérivé un type d'inégalités valides basées sur la fonction de coût. Ils ont obtenu de bons résultats pour des instances de taille moyenne.

Hoang [45] a accéléré l'algorithme de Branch-and-Bound grâce à l'amélioration de la borne inférieure, et ce en résolvant le problème du plus court chemin obtenu suie à la relaxation des contraintes d'intégralité, et en minimisant sur l'ensemble défini par la contrainte de budget.

Dionne et Florian [25] ont amélioré la méthode proposée par Hoang [45] en utilisant une règle de branchement sur la variable qui induit la meilleure amélioration de l'objectif par unité de budget.

Gallo [32] a présenté un autre algorithme de Branch-and-Bound pour résoudre le problème avec la contrainte de budget. Cet algorithme utilise de nouvelles bornes supérieures et inférieures qui sont meilleures que celles utilisées dans les algorithmes précédents.

Lamar et al. [57] ont proposé un schéma de linéarisation interactive intégré dans un algorithme de Branch-and-Bound. Cet algorithme a été appliqué avec succès sur un problème de conception de réseaux avec des coûts fixes, comme c'est le cas du problème de planification de la charge de transporteur routier (*load planning problem of less-than-truckload, LTL, motor carriers*).

Hellstrand et al.[42] ont réalisé une étude polyédrale du MUND afin de dériver de nouvelles inégalités valides pertinentes qui améliorent les relaxations du problème. Ils

ont démontré que le polytope de la relaxation linéaire de la formulation forte possède la propriété d'être quasi-entier. Enfin, des différentes relaxations et méthodes de décomposition ont été discutées par Migdalas [63].

Bien que la plupart des algorithmes proposés peuvent réussir à résoudre les problèmes de petite à moyenne tailles, leur temps de calcul croît rapidement avec la taille du problème.

3.1.4.2 Méthodes heuristiques

Étant donné la difficulté du MUND, plusieurs chercheurs se sont intéressés aux approches heuristiques pour le résoudre.

Kratica et al. [55] ont proposé une heuristique basée sur l'algorithme génétique (GA), qui performe mieux sur les instances de taille moyenne. Lakshmi [56] a aussi appliqué l'algorithme génétique sur le problème MUND, mais les résultats sont dominés par la méthode de duale-ascendante combinée à l'heuristique de suppression-ajout (*drop-add heuristic*).

Wong [72] a proposé une heuristique basée sur le lien d'inclusion (*link inclusion heuristic procedure*) pour le problème sans capacité avec la contrainte du budget.

Plusieurs heuristiques classiques basées sur les heuristiques d'ajout (*add heuristic*), de suppression (*drop heuristic*), et de suppression-ajout (*drop-add heuristic*), ainsi que sur les heuristiques qui combinent ces trois types ont été proposées pour résoudre le problème MUND sans et avec contraintes de budget. Nous citons : Boffey et Hinxman [11], Los et Lardinois [60], Billheimer et Gray [10], Dionne et Florian [25] et Wong [71].

Le lecteur est invité à consulter l'article de Magnanti et Wong [61], où ils ont présenté d'autres méthodes de résolution du problème MUND.

3.2 Conception de réseaux avec coûts fixes et capacités

Le problème de conception de réseaux avec coûts fixes et capacités MCND (Multicommidity Capacitated Network Design) est une généralisation du problème MUND

présenté dans la section 3.1, où une capacité u_{ij} est imposée sur les arcs (i, j) telle que $u_{ij} \leq \sum_{k \in K} d^k$. Notons que le problème MUND est obtenu lorsque $u_{ij} = \sum_{k \in K} d^k$.

Étant donné que le problème MUND est un cas particulier du MCND, et qu'il est un NP-difficile, le problème MCND fait partie lui aussi de la classe NP-difficile. Il pose ainsi d'importants défis de recherche, notamment le fait que les contraintes de capacités lient les flots des différents produits sur chaque arc.

Comme pour le problème MUND, la fonction objectif du problème MCND consiste également à minimiser les coûts de transport des produits et les coûts de conception des arcs dans le réseau, tout en respectant la contrainte de capacité imposée sur chaque arc. On retrouve deux types de coûts déjà présentés dans la section (4.3.1) :
- c_{ij}^k, qui représente le coût unitaire de transport du produit k sur l'arc (i, j) ;
- f_{ij}, qui est un coût imposé si l'arc (i, j) est considéré dans le réseau.
De plus, les variables x_{ij}^k représentent la quantité de flot du produit k qui circule sur l'arc (i, j), tandis que les variables y_{ij} sont des variables binaires qui prennent la valeur 1 si l'arc (i, j) est parcouru et la valeur 0 sinon.

La formulation faible de ce problème s'écrit comme suit [16] :

$$\min \quad \sum_{k \in K} \sum_{(i,j) \in A} c_{ij}^k x_{ij}^k + \sum_{(i,j) \in A} f_{ij} y_{ij} \tag{3.28}$$

Sujet à

$$\sum_{j \in V_i^+} x_{ij}^k - \sum_{j \in V_i^-} x_{ji}^k = \begin{cases} d^k, & \text{si } i = O(k), \\ -d^k, & \text{si } i = D(k), \\ 0, & \text{sinon.} \end{cases} \qquad \forall i \in V, k \in K, \tag{3.29}$$

$$\sum_{k \in K} x_{ij}^k \leq u_{ij} y_{ij}, \qquad \forall (i, j) \in A, \tag{3.30}$$

$$x_{ij}^k \geq 0, \qquad \forall (i, j) \in A, k \in K, \tag{3.31}$$

$$y_{ij} \in \{0,1\}, \qquad \forall (i,j) \in A. \tag{3.32}$$

La formulation forte est obtenue par l'introduction des inégalités valides de type :

$$x_{ij}^k \leq d^k y_{ij}, \qquad \forall (i,j) \in A, k \in K, \tag{3.33}$$

dans le modèle. Ces inégalités améliorent la qualité de la borne inférieure de la relaxation linéaire qu'elles fournissent. Notons que dans ce cas, il n'est pas possible de désagréger les contraintes de capacités (3.29) puisque la condition $u_{ij} = \sum_{k \in K} d^k$, n'est plus vérifiée.

La formulation forte s'écrit sous la forme suivante [33] :

$$\min \quad \sum_{k \in K} \sum_{(i,j) \in A} c_{ij}^k x_{ij}^k + \sum_{(i,j) \in A} f_{ij} y_{ij} \tag{3.34}$$

Sujet à

$$\sum_{j \in V_i^+} x_{ij}^k - \sum_{j \in V_i^-} x_{ji}^k = \begin{cases} d, & \text{si } i = O(k), \\ -d, & \text{si } i = D(k), \\ 0, & \text{sinon.} \end{cases} \qquad \forall i \in V, k \in K, \tag{3.35}$$

$$\sum_{k \in K} x_{ij}^k \leq u_{ij} y_{ij}, \qquad \forall (i,j) \in A, \tag{3.36}$$

$$x_{ij}^k \leq d^k y_{ij}, \qquad \forall (i,j) \in A, k \in K, \tag{3.37}$$

$$x_{ij}^k \geq 0, \qquad \forall (i,j) \in A, k \in K, \tag{3.38}$$

$$y_{ij} \in \{0,1\}, \qquad \forall (i,j) \in A. \tag{3.39}$$

3.2.1 Cas particuliers

Plusieurs modèles apparentés au problème MCND, et qui représentent des cas particuliers, ont été largement étudiés dans la littérature. Parmi ceux-ci, on retrouve le problème de localisation avec capacité (Gendron et Crainic [36], Van Roy [70]), le problème de voyageur de commerce (Fischtti et Toth [30]), le problème de multiflot à coût minimum (Barnhart [7]). Pour une revue complète et détaillée des différents modèles apparentés au problème MCND, le lecteur est invité à consulter les articles de Magnanti et Wong [61] et de Minoux [64].

3.2.2 Revue de littérature pour le problème de conception de réseaux avec coûts fixes et capacités

Plusieurs approches ont été envisagées pour résoudre le problème de conception de réseaux avec coûts fixes et capacités. Nous présentons dans cette partie quelques approches exactes et heuristiques qui ont été développées pour résoudre le problème MCND.

3.2.2.1 Méthodes exactes

Les méthodes exactes appliquées pour résoudre le problème avec capacités se basent généralement sur les méthodes de coupes, la relaxation lagrangienne et la décomposition de Benders.

Holmberg et Yuan [48] ont développé une méthode exacte basée sur une relaxation lagrangienne pour obtenir des sous-problèmes faciles à résoudre. Le dual lagrangien est résolu par la méthode du sous-gradient. L'heuristique lagrangienne est ensuite intégrée dans un algorithme de Branch-and Bound. La méthode a généré de bonnes solutions réalisables aux problèmes de grande taille dans un délai raisonnable. Cet algorithme a été par la suite modifié par Kliewer et al. [53], qui y ont intégré une méthode de génération de coupes.

D'autres approches basées également sur la relaxation lagrangienne combinée à la méthode de Branch-and-Bound ont été abordées par Gendron et Crainic [37], Gendron

et al. [38] et Crainic et al. [21]. Gendron et Crainic [35] ont démontré que différentes relaxations du problème fournissent la même borne que celle fournie par la relaxation linéaire de la formulation forte. Crainic et al. [21] ont comparé par la suite les méthodes de faisceau et de sous-gradient appliquées à deux types de relaxation lagrangienne.

Costa et al. [17] ont appliqué la décomposition de Benders au problème. Les résultats ont démontré l'efficacité de la méthode sur des instances de petite taille. Chouman et al. [14] ont proposé une méthode de coupes afin de calculer de meilleures bornes inférieures, et ce en testant différents types d'inégalités valides.

L'ajout des inégalités valides pour améliorer la formulation parait important pour résoudre le problème avec capacités en raison de la difficulté à le résoudre par les méthodes traditionnelles de programmation linéaire [21]. Plusieurs chercheurs ont tenté de développer des approches et d'étudier des formulations intégrant des inégalités valides, en effectuant des analyses polyédrales, en particulier, Chouman et al. [16], Atamtürk [3], Atamtürk et Rajan [5] Atamtürk et Günlük [4], Bienstock et Günlük [9].

Plus récemment, Gendron et Larose [39] ont développé une méthode de Branch-and-Price-and-Cut pour résoudre le problème MCND. De bons résultats ont été obtenus notamment pour les instances de grande taille. Chouman et al.[15] ont obtenu également de bons résultats, après voir appliqué l'algorithme de Branch-and-Bound combiné avec la méthode de génération de coupes, en intégrant des méthodes de filtrage basées sur des arguments de dualité ou sur la détection de solutions non réalisables.

3.2.2.2 Méthodes heuristiques

Étant donné la difficulté du problème MCND, plusieurs chercheurs ont proposé des approches heuristiques afin d'obtenir une meilleure borne supérieure sur la valeur optimale du problème.

Crainic et al. [19] ont proposé une métaheuristique basée sur la recherche avec tabous, l'algorithme du simplexe et la génération de colonnes. Une version parallèle basée sur cette approche a été développée par Crainic et Gendreau [22].

Ghamlouche et al. [40] ont développé une nouvelle structure de voisinage à base de cycle pour la recherche avec tabous. Les résultats expérimentaux ont montré que

l'approche proposée était performante. Ils ont ensuite utilisé ce voisinage à base de cycle dans un algorithme de *path-relinking* [41]. Une version parallèle basée sur la recherche avec tabous a été développée par Crainic et al. [20].

Une heuristique basée sur une descente locale a été proposée par Powell[66]. Tourillon [69] a développé la méthode de duale-ascendante, tandis que Hernu [43] a proposé une heuristique basée sur des méthodes de programmation mathématique.

Chouman et Crainic [13] ont développé une métaheuristique combinant la recherche avec tabous avec une méthode de génération de coupes et un algorithme de Branch-and-Bound.

Katayama et al. [50] ont combiné le « capacity scaling » et le branchement local en intégrant la méthode de génération de colonnes et de coupes. Elle est considérée parmi les méthodes heuristiques les plus efficaces avec un écart d'optimalité moyen de moins de 1%.

D'autres méthodes heuristiques combinées à des techniques de programmation linéaire en nombres entiers ont été proposées. Nous en citons : Kim et Barnhart [51], Kim et al. [52], Fischetti et Lodi [29] et Rodriguez-Martin et Salazar-Gonzalez [67].

Nous invitons le lecteur intéressé par une revue de littérature plus approfondie à consulter les références suivantes : Crainic [18], El Filali [27], Frangioni et Gendron [31], Gendron [34], Gendron et al. [38], Katayama et al. [50] et Larose [58], références sur lesquelles nous nous sommes basé pour fournir un aperçu de la littérature existante du problème de conception de réseaux avec coûts fixes et capacités.

CHAPITRE 4

ALGORITHME DE BRANCH-AND-PRICE-AND-CUT

Nous décrivons dans ce chapitre les différentes étapes de notre algorithme, qui consiste à intégrer les méthodes de génération de colonnes et de génération de coupes dans l'algorithme de Branch-and-Bound pour résoudre le problème de conception de réseaux avec coûts fixes et sans capacité. Notons que cette méthode s'inspire d'une approche similaire proposée par Gendron et Larose [39] pour les problèmes avec coûts fixes et capacités.

4.1 Solution initiale

L'algorithme de Branch-and-Bound parcourt l'arbre de recherche pour trouver une solution optimale, ou déterminer une meilleure solution réalisable, si le temps de la résolution atteint sa limite. L'algorithme augmente en efficacité en optimisant le temps de calcul s'il tire profit d'une bonne borne supérieure qui lui permet d'élaguer, le plus tôt possible, des nœuds qui sont inintéressants dans l'arbre de recherche.
À cet effet, la meilleure borne supérieure obtenue dans un temps limite de trois heures par l'une de ces méthodes : Branch-and-Bound, Branch-and-Cut, et Branch-and-Price-and-Cut qui représente généralement la solution optimale, est considérée la borne supérieure initiale (solution réalisable initiale). Le rôle de notre algorithme revient à prouver que cette borne supérieure est la solution optimale, ou de l'améliorer légèrement dans le cas contraire, afin d'avoir une solution optimale du problème à résoudre.

4.2 Évaluation d'un nœud

À chaque nœud de l'arbre de recherche, nous résolvons la relaxation linéaire en utilisant une technique qui alterne entre la résolution du sous-problème d'évaluation qui permet de générer de nouvelles colonnes, et la résolution du sous-problème de séparation qui génère de nouvelles inégalités valides, qui conduisent principalement à une

amélioration de la borne inférieure et une réduction du temps de calcul.
Nous présentons dans la section suivante les différentes étapes de l'évaluation d'un nœud.

4.2.1 Relaxation linéaire

Dans un problème de minimisation, la relaxation linéaire fournit une borne inférieure sur la valeur optimale du problème linéaire en nombres entiers. Cette borne est moins bonne si la relaxation linéaire est appliquée sur la formulation faible du problème, puisqu'elle donne un écart d'intégralité plus grand, ce qui augmente le temps d'exécution de l'algorithme, raison pour laquelle nous avons choisi la formulation forte Arc-Produit présentée dans la section 3.1.2.2.
La relaxation linéaire du problème MUND s'écrit comme suit :

$$\min \quad \sum_{k\in K}\sum_{(i,j)\in A} C_{ij}^k x_{ij}^k + \sum_{(i,j)\in A} f_{ij} y_{ij} \tag{4.1}$$

Sujet à

$$\sum_{j\in V_i^+} x_{ij}^k - \sum_{j\in V_i^-} x_{ji}^k = \begin{cases} 1, & \text{si } i = O(k), \\ -1, & \text{si } i = D(k), \\ 0, & \text{sinon.} \end{cases} \qquad \forall i \in V, k \in K, \tag{4.2}$$

$$x_{ij}^k \leq y_{ij}, \qquad \forall (i,j) \in A, k \in K, \tag{4.3}$$

$$y_{ij} \leq 1, \qquad \forall (i,j) \in A, \tag{4.4}$$

$$x_{ij}^k \geq 0, \qquad \forall (i,j) \in A, k \in K, \tag{4.5}$$

$$y_{ij} \geq 0, \qquad \forall (i,j) \in A. \tag{4.6}$$

Cependant, la contrainte (4.4) n'est plus nécessaire dans ce programme, étant donné

qu'il s'agit d'un problème de minimisation, ce qui va pousser les variables y_{ij} à leurs valeurs minimales dans les contraintes (4.3). Comme les variables x_{ij}^k prennent des valeurs inférieures à 1, les variables y_{ij} le seront aussi nécessairement. Ce qui implique que la valeur maximale que va prendre la variable y_{ij} est la valeur 1, et par conséquent, la relaxation linéaire peut prendre la forme suivante, en associant des variables duales aux contraintes :

$$\min \quad \sum_{k\in K}\sum_{(i,j)\in A} C_{ij}^k x_{ij}^k + \sum_{(i,j)\in A} f_{ij} y_{ij} \tag{4.7}$$

Sujet à

$$\sum_{j\in V_i^+} x_{ij}^k - \sum_{j\in V_i^-} x_{ji}^k = \begin{cases} 1, & \text{si } i = O(k), \\ -1, & \text{si } i = D(k), \\ 0, & \text{sinon.} \end{cases} \qquad \forall i \in V, k \in K, \qquad (\pi_i^k) \tag{4.8}$$

$$x_{ij}^k \leq y_{ij}, \qquad \forall (i,j) \in A, k \in K, \qquad (\alpha_{ij}^k \geq 0) \tag{4.9}$$

$$x_{ij}^k \geq 0, \qquad \forall (i,j) \in A, k \in K, \tag{4.10}$$

$$y_{ij} \geq 0, \qquad \forall (i,j) \in A. \tag{4.11}$$

Pour résoudre la relaxation linéaire de notre problème MUND, nous appliquons la méthode de génération de colonnes aux variables de flot x. Nous utilisons par la suite la génération de coupes pour introduire les inégalités fortes violées (4.9), afin d'améliorer la borne inférieure sur la valeur optimale du problème initial en nombres entiers obtenue par la génération de colonnes. Nous faisons appel aussi à la technique de fixation des variables basée sur les coûts réduits des variables de conception y, afin d'améliorer davantage la performance de l'algorithme de Branch-and-Price-and-Cut. Une autre technique est adoptée pour accélérer ce processus, il s'agit de suspendre les méthodes de génération de colonnes et de génération de coupes avant d'atteindre l'optimalité de la

relaxation linéaire en se basant sur des critères de convergence.

Nous décrivons en détail toutes ces méthodes dans les sections qui suivent.

4.2.2 Génération de colonnes

La méthode de génération de colonnes est appliquée sur un modèle ayant un très grand nombre de variables, généralement obtenu après une reformulation du problème original, ce qui rend difficile de le résoudre par l'algorithme du simplexe. La méthode résout itérativement un ou plusieurs problèmes restreints, ainsi que plusieurs sous-problèmes. Elle débute avec un sous ensemble de variables, et à chaque itération, elle ajoute des variables pouvant améliorer la solution courante du problème maître. Dans notre cas, la méthode de génération de colonnes est appliquée sur les variables de flot x_{ij}^k, pour résoudre la relaxation linéaire de la formulation forte du problème MUND.
Les différents composants de la méthode sont présentés dans la partie qui suit.

4.2.2.1 Problème maître

Dans notre cas, il n'y a aucune reformulation du problème original MUND. En effet, le problème maître correspond à la relaxation linéaire de la formulation forte du problème MUND comme présentée dans la section 4.2.1.

4.2.2.2 Problème maître restreint

Pour identifier le problème maître restreint (PMR), nous choisissons seulement un sous ensemble des variables de flot x_{ij}^k du problème maître, certaines variables de flot sont restreintes à être nulles.
Nous élargissons l'ensemble des arcs avec des arcs artificiels reliant $O(k)$ à $D(k)$ pour chaque produit k, ces arcs sont sans capacité, ils n'ont pas de coûts de conception $f_{O(k)D(k)} = 0$, et ils possèdent un coût de transport très important, qu'on définit ainsi :

$$C^k_{O(k)D(k)} = \sum_{k \in K} \sum_{(i,j) \in A} C^k_{ij} + \sum_{(i,j) \in A} f_{ij}.$$

En ajoutant ces arcs, nous nous assurons non seulement que chaque problème maître restreint est toujours réalisable, mais ceci nous permettra principalement d'obtenir la

première solution réalisable pour lancer la génération de colonnes.
En outre, si la solution actuelle du PMR comporte au moins un arc artificiel, la valeur de cette solution sera très grande, étant donné le coût très élevé de l'arc artificiel. Par conséquent, cette solution sera éliminée dans les premières itérations de la méthode (sauf si le problème relaxé n'est pas réalisable).
Pour définir le problème maître restreint, on associe à chaque arc $(i,j) \in A^+$ un sous ensemble de produits $\tilde{K} \subseteq K$, où A^+ définit l'ensemble de tous les arcs $(i,j) \in A$, ainsi que les arcs artificiels : $A^+ = A \bigcup \{(O(k), D(k)), \forall k \in K\}$.
On définit l'ensemble $\tilde{A}^+$, tel que $\tilde{A}^+ = \{(i,j) \in A^+ | k \in \tilde{K}\}$, et on dénote par :
$\tilde{V}_i^+ = \{j \in V | (i,j) \in \tilde{A}^+\}$ et $\tilde{V}_i^- = \{j \in V | (j,i) \in \tilde{A}^+\}$.
On dénote par $\tilde{\tilde{K}}$, ($\tilde{\tilde{K}} \subseteq \tilde{K}$), le sous ensemble d'inégalités valides déjà générées dans l'ensemble $\tilde{K}$, i.e., les inégalités valides fortes (4.9).

Le problème maître restreint est écrit sous la forme suivante :

$$\min \quad \sum_{k \in \tilde{K}} \sum_{(i,j) \in A^+} C_{ij}^k x_{ij}^k + \sum_{(i,j) \in A^+} f_{ij} y_{ij} \tag{4.12}$$

Sujet à

$$\sum_{j \in \tilde{V}_i^+} x_{ij}^k - \sum_{j \in \tilde{V}_i^-} x_{ji}^k = \begin{cases} 1, & \text{si } i = O(k), \\ -1, & \text{si } i = D(k), \forall i \in V, k \in \tilde{K}, \\ 0, & \text{sinon}, \end{cases} \tag{4.13}$$

$$x_{ij}^k \leq y_{ij}, \qquad \forall (i,j) \in A^+, k \in \tilde{\tilde{K}} \subseteq \tilde{K}, \tag{4.14}$$

$$x_{ij}^k \geq 0, \qquad \forall (i,j) \in A^+, k \in \tilde{K}, \tag{4.15}$$

$$y_{ij} \geq 0, \qquad \forall (i,j) \in A^+. \tag{4.16}$$

La formulation initiale du problème maître restreint est obtenue en n'utilisant que les variables associées aux arcs artificiels. Après avoir résolu le PMR, de nouvelles colonnes

(s'il y a lieu) sont ajoutées itérativement à ce dernier. Étant donné que l'ajout d'une variable ne change pas complètement la solution en général, il est donc préférable, de ne pas relancer l'algorithme primal du simplexe à chaque itération, mais plutôt de le reexécuter en partant de l'ancienne solution pour en avoir une nouvelle, étant donné que l'ancienne solution demeure toujours réalisable pour le nouveau problème maître restreint.

4.2.2.3 Sous-problème

Le sous-problème consiste à identifier les variables de flot x_{ij}^k qui ne sont pas encore générées dans le problème maître restreint, et qui peuvent améliorer la solution optimale du problème maître. En fait, le sous-problème calcule les coûts réduits des variables de flot x_{ij}^k, $(i,j) \in A$, $k \notin \tilde{K}$ à partir du dual du problème maître restreint.
Le dual de la relaxation linéaire du problème original s'écrit sous la forme suivante :

$$\max \quad \sum_{k \in K} (\pi_{O(k)}^k - \pi_{D(k)}^k) \tag{4.17}$$

Sujet à

$$\pi_i^k - \pi_j^k - \alpha_{ij}^k \leq C_{ij}^k, \qquad \forall (i,j) \in A, k \in K, \qquad (x_{ij}^k \geq 0) \tag{4.18}$$

$$\sum_{k \in K} \alpha_{ij}^k \leq f_{ij}, \qquad \forall (i,j) \in A, \qquad (y_{ij} \geq 0) \tag{4.19}$$

$$\alpha_{ij}^k \geq 0, \qquad \forall (i,j) \in A, k \in K, \tag{4.20}$$

Nous déduisons par la contrainte (4.18) la formule des coûts réduits des variables x_{ij}^k :

$$C_{ij}^k - \pi_i^k + \pi_j^k + \alpha_{ij}^k, \qquad \forall (i,j) \in A, k \in K$$

Seulement les variables de flot qui ont des coûts réduits négatifs peuvent améliorer la solution optimale du problème maître, c'est-à-dire celles qui satisfont :

$$C_{ij}^k - \pi_i^k + \pi_j^k + \alpha_{ij}^k < 0, \qquad \forall (i,j) \in A, k \in K.$$

Les variables duales π_i^k sont connues après avoir résolu le problème maître restreint, tandis que les variables duales α_{ij}^k associées aux contraintes (4.14) ne le sont pas complètement, vu que les contraintes ne sont pas totalement générées par la génération de coupes, qui est appliquée, rappelons-le, aux contraintes $x_{ij}^k \leq y_{ij}$, $\forall (i,j) \in A^+, k \in K$. Pour les calculer, nous nous basons sur les équations d'écarts complémentaires définies comme suit :

$$x_{ij}^k (C_{ij}^k - \pi_i^k + \pi_j^k + \alpha_{ij}^k) = 0, \qquad \forall (i,j) \in A, k \in K, \tag{4.21}$$

$$y_{ij} (f_{ij} - \sum_{k \in K} \alpha_{ij}^k) = 0, \qquad \forall (i,j) \in A, \tag{4.22}$$

$$\alpha_{ij}^k (y_{ij} - x_{ij}^k) = 0, \qquad \forall (i,j) \in A, k \in K. \tag{4.23}$$

Pour avoir une solution optimale de la relaxation linéaire, qui est le problème maître (PM), il faut que toutes les égalités de (4.21) à (4.23) soient satisfaites. Cependant, si $k \in \tilde{K}$, alors toutes ces contraintes sont satisfaites puisque le problème maître restreint est résolu à l'optimum.
Notre but est alors d'identifier les variables de flot x_{ij}^k qui ne satisfont pas les conditions d'optimalité du coût réduit et qui n'appartiennent pas à $\tilde{K}$. Pour cela, on suppose que $(\widehat{x}, \widehat{y})$ est la solution optimale du PMR, et $(\widehat{\pi}, \widehat{\alpha})$ celle du dual du PMR.

Pour $k \notin \tilde{K}$, pour chaque arc $(i,j) \in A$, nous distinguons deux cas, selon que les variables y_{ij} sont positives ou nulles :

- Cas 1 : $\widehat{y}_{ij} > 0$.

  Pour que la solution du problème maître restreint soit optimale pour la relaxation linéaire du problème maître original (MUND), il faut que la contrainte d'écarts complémentaires (4.23) soit satisfaite :

$$\widehat{\alpha}_{ij}^k(\underbrace{\widehat{y}_{ij}}_{>0} - \underbrace{\widehat{x}_{ij}^k}_{=0}) = 0 \qquad \Rightarrow \qquad \widehat{\alpha}_{ij}^k = 0$$

Ce qui implique que la contrainte d'optimalité du coût réduit des variables de flot x_{ij}^k pour $k \notin \tilde{K}$ (4.18) devient :

$$C_{ij}^k - \pi_i^k + \pi_j^k \geq 0.$$

Seules les variables de flot dont les coûts réduits sont négatifs sont alors ajoutées au problème maître restreint :

$$C_{ij}^k - \pi_i^k + \pi_j^k < 0.$$

- Cas 2 : $\widehat{y}_{ij} = 0$.
  Si $\widehat{y}_{ij} = 0$, alors $\widehat{x}_{ij}^k = 0$, $\forall k \in K$ (la contrainte (4.9) impose un flot nul si l'arc n'est pas conçu).
  Dans ce cas, par la contrainte (4.18) du dual, nous avons :

$$\alpha_{ij}^k \geq \pi_i^k - \pi_j^k - C_{ij}^k. \tag{4.24}$$

Nous combinons les contraintes (4.20) ($\alpha_{ij}^k \geq 0$) et (4.24), nous obtenons l'inégalité suivante :

$$\alpha_{ij}^k \geq max(0, \pi_i^k - \pi_j^k - C_{ij}^k). \tag{4.25}$$

De plus, nous avons la condition d'optimalité du coût réduit de la variable y_{ij} (4.19) :

$$f_{ij} \geq \sum_{k \in K} \alpha_{ij}^k, \qquad \forall (i,j) \in A. \tag{4.26}$$

À partir des contraintes (4.25) et (4.26), nous obtenons :

$$f_{ij} \geq \sum_{k \in K} max(0, \pi_i^k - \pi_j^k - C_{ij}^k). \tag{4.27}$$

Si la solution du problème maître restreint est optimale pour le problème maître, alors la contrainte d'optimalité (4.27) est satisfaite. Dans le cas contraire, on ajoute les variables des flot x_{ij}^k qui ne satisfont pas cette inégalité, et dont les coûts réduits sont négatifs, c'est-à-dire, telles que $C_{ij}^k - \pi_i^k + \pi_j^k < 0$, pour $k \notin \tilde{K}$ seulement.

En résumé, pour générer les variables de flot x_{ij}^k améliorant la solution optimale du problème maître, on distingue deux cas :

1. Si $y_{ij} > 0$ et $C_{ij}^k - \pi_i^k + \pi_j^k < 0$, $k \notin \tilde{k}$, alors on ajoute les variables x_{ij}^k au PMR.
2. Si $y_{ij} = 0$, et $f_{ij} < \sum_{k \in K} \max\ (0, \pi_i^k - \pi_j^k - C_{ij}^k)$, alors pour tout $k \notin \tilde{k}$, tel que $C_{ij}^k - \pi_i^k + \pi_j^k < 0$, les variables x_{ij}^k sont ajoutées au PMR.

Le processus d'ajout de variables au PMR, puis de résolution du nouveau PMR se poursuit, jusqu'à atteindre l'optimalité du problème maître (la relaxation linéaire). Une fois la génération de colonnes est terminée, nous obtenons une borne inférieure Z_{RL} sur la valeur optimale du problème MUND. Si Z_{RL} est entière et inférieure à la meilleure solution réalisable obtenue par l'algorithme de Branch-and-Bound, alors la solution Z_{RL} devient la meilleure solution réalisable du MUND. Si par contre, Z_{RL} est supérieure à la meilleure solution réalisable du MUND, le nœud courant est directement élagué sans passer à la génération de coupes. Lorsqu'aucun de ces deux cas ne se présente, nous appliquons la génération de coupes pour améliorer la borne inférieure Z_{RL}.

4.2.3 Génération de coupes

Quand le processus de la génération de colonnes se termine, il est temps de vérifier si toutes les inégalités valides définies par (4.28) sont satisfaites par la solution optimale du problème maître obtenue par la génération de colonnes.

$$x_{ij}^k \leq y_{ij} \qquad \forall (i,j) \in A, \forall k \in K. \tag{4.28}$$

La génération de coupes débute sans aucune inégalité valide, celles qui sont violées par la solution optimale du problème maître sont ajoutées au fur et à mesure au problème, jusqu'à ce que aucune inégalité valide ne soit violée.

Étant donné que le nombre des inégalités valides est important, ces dernières sont ajoutées dynamiquement au problème pour ne pas ralentir l'algorithme d'une part, et pour éviter la dégénérescence, d'autre part.
Rappelons que les inégalités valides identifient la formulation forte, et elles permettent à la relaxation linéaire de fournir un écart d'intégralité réduit, ce qui permet à l'algorithme de Branch-and-Bound d'élaguer plus de nœuds, et qui influence positivement le temps de résolution.

S'il existe des inégalités valides violées par la solution optimale du problème maître, celles ci sont ajoutées au problème maître, et ce dernier est résolu à nouveau. La solution actuelle n'est plus alors optimale, par contre, elle demeure toujours réalisable pour le problème dual du nouveau problème, puisque l'ajout d'une contrainte au problème primal est équivalent à l'ajout d'une colonne à son dual. La nouvelle formulation obtenue est alors résolue par l'algorithme dual du simplexe, en partant de la dernière solution optimale du problème maître comme solution de base. La méthode se termine lorsqu'il n'y a aucune inégalité valide violée par la solution optimale obtenue.

Après l'ajout des inégalités valides, et avant de reéxécuter la génération de colonnes à nouveau, on fait appel à la technique de fixation de variables présentée dans la section suivante.

Ce processus se déroule alternativement jusqu'au moment où il n'y a aucune variable de flot, ni une inégalité valide, à ajouter au problème maître. La solution obtenue à la fin de ce processus est la solution optimale de la relaxation linéaire.

4.2.4 Fixation de variables

La technique de fixation de variables est appliquée afin d'accélérer l'algorithme de Branch-and-Price-and-Cut (B&P&C) en réduisant le domaine de variables. Elle consiste à éliminer des variables de conception y_{ij} qui n'améliorent pas la valeur optimale du problème quand on les introduit avec une valeur égale à un moins sa valeur hors-base [65].

Elle est basée sur les coûts réduits des variables de conception, une borne inférieure et une borne supérieure sur la valeur optimale du problème, afin d'évaluer $Z_{RL} + |\overline{f}_{ij}|$ qui définit une borne inférieure du nouveau problème, où $\overline{f}_{ij} = f_{ij} - \sum_{k \in K} \overline{\alpha}_{ij}^{k}$ (contrainte (4.19)).

Soient Z_{RL} la valeur optimale de la relaxation linéaire qui représente une borne inférieure sur la valeur optimale du problème MUND, $\overline{y}_{ij}$ sa solution optimale, et Z_{CD} la solution candidate qui représente une borne supérieure sur la valeur optimale du MUND. À chaque itération de génération de coupes, nous vérifions pour chaque arc (i, j) si $Z_{RL} + |\overline{f}_{ij}| \geq Z_{CD}$. Si c'est le cas, alors on attribue la valeur de $\overline{y}_{ij}$ à y_{ij}. Plus précisément, si $\overline{y}_{ij} = 0$, alors $y_{ij} = 0$, et si $\overline{y}_{ij} = 1$, alors $y_{ij} = 1$. Par contre pour $0 < \overline{y}_{ij} < 1$, et à partir des équations des écarts complémentaires (4.19) et (4.22), on obtient $\overline{f}_{ij} = 0$, cela signifie que $Z_{RL} + |\overline{f}_{ij}| = Z_{RL} < Z_{CD}$, et par conséquent le nouveau problème ne génère pas une solution meilleure si on introduit la variable de conception y_{ij}.

Pour une démonstration détaillée, le lecteur est invité à consulter l'article [39].

4.2.5 Critère d'arrêt

Pour accélérer le processus d'évaluation d'un nœud donné dans l'arbre de recherche, une approche basée sur la notion de convergence de la valeur optimale de la relaxation linéaire obtenue par la génération de colonnes et la génération de coupes alternativement est adoptée, afin de réduire le temps de calcul de la relaxation linéaire et de l'algorithme de Branch-and-Price-and-Cut.

À la fin de chaque méthode, nous comparons la valeur optimale obtenue soit par la génération de colonnes ou bien par la génération de coupes avec les valeurs antécédentes. Si on constate que cette valeur ne s'améliore pas, c'est-à-dire, si l'écart est proche d'une valeur ε pendant un nombre d'itérations τ, le processus de l'évaluation du nœud s'arrête en obtenant une solution approchée, et en s'assurant qu'elle représente toujours une borne inférieure sur la valeur optimale du problème MUND.

L'algorithme de l'évaluation du nœud converge alors vers une solution réalisable du problème relaxé, probablement non optimale mais proche de l'optimum.

Le nombre d'itérations τ et le taux d'approximation ε sont choisis après plusieurs tests effectués sur plusieurs instances de différentes tailles (pour plus de détail voir le chapitre 5).

Après avoir présenté la manière dont est évaluée la solution optimale d'un nœud sélectionné, nous présentons dans ce qui suit l'approche adoptée pour choisir le prochain nœud à évaluer, ainsi que la variable de branchement si le cas le nécessite (i.e., si $Z_{RL} < Z_{CD}$ et y fractionnaire).

4.3 Règle d'exploration de l'arbre

La sélection du prochain nœud peut être réalisée de diverses façons. Afin de bien analyser la méthode appliquée dans notre travail, deux stratégies de branchement sont testées : la stratégie meilleur d'abord et la stratégie en profondeur. Par contre, la recherche hybride est éliminée puisqu'elle ne donne pas de bons résultats.

4.4 Règle de branchement

Le branchement fiable est la règle choisie dans ce travail, puisqu'elle est jugée dans la littérature parmi les règles de branchement les plus efficaces décrites à ce jour [2, 15, 58].

4.5 Algorithme de Branch-and-Price-and-Cut

Nous résumons dans cette partie les différentes étapes de la méthode de Branch-and-Price-and-Cut (B&P&C), l'algorithme est présenté dans la Figure 4.1 :

Entrée : problème de minimisation P.
Sortie : x^*, y^* solutions optimales, Z^* valeur optimale.

1. Initialisation :
    - Z^* : une solution réalisable initiale.

- Z_{Prec} : une borne inférieure initialisée à 0.
- $\Gamma = \{P\}$: l'ensemble des nœuds à évaluer.
- $\varphi = 0$: un compteur.
- max : un nombre maximum d'itérations.
- ξ : l'écart de convergence (l'écart entre la borne inférieure obtenue et sa précédente).

2. Sélection :
   - Sélectionner un nœud $R \in \Gamma$ à évaluer, et $\Gamma \leftarrow \Gamma \backslash \{R\}$.
3. Évaluation : Calcul d'une borne inférieure Z_{RL} en résolvant la relaxation linéaire de R :
   (a) Trouver une solution primale-duale $(\bar{y}_{ij}, \bar{x}^k_{ij})$ et $(\overline{\alpha}^k_{ij}, \overline{\beta}_{ij})$ respectivement pour le PMR initial en utilisant l'algorithme dual du simplexe.
   Génération de colonnes :
   (b) Pricing : pour chaque arc $(i, j) \in A$, si :
   $\bar{y}_{ij} > 0$ ou $(\bar{y}_{ij} = 0$ et $f_{ij} \geq \sum_{k \in K} \max\ (0, \pi^k_i - \pi^k_j - C^k_{ij}))$, alors, $\forall k \notin \tilde{K}$, tel que $(C^k_{ij} - \pi^k_i + \pi^k_j < 0)$, ajouter la variable de flot x^k_{ij} au PMR.
   (c) Si x^k_{ij} est ajouté au PMR, résoudre le nouveau PMR par l'algorithme primal du simplexe pour obtenir une solution primale-duale $(\bar{y}_{ij}, \bar{x}^k_{ij})$ et $(\overline{\alpha}^k_{ij}, \overline{\beta}_{ij})$ respectivement et aller à l'étape 3b.
   (d) Soit Z_{RL} la borne inférieure obtenue par la génération de colonnes :
      i. Si $Z_{RL} < Z^*$, et $\bar{y}$ entier, alors $Z^* = Z_{RL}$, $x^* = \bar{x}$ et $y^* = \bar{y}$.
      ii. Si $Z_{RL} \geq Z^*$, aller à l'étape 5.
      iii. Si $((Z_{RL} - Z_{Prec})/Z_{RL}) < \xi$, $++\varphi$, sinon $\varphi = 0$.
      $Z_{Prec} = Z_{RL}$.
      iv. Si ($\varphi = max$) , alors aller à l'étape 5.

   Génération de coupes :

(e) Séparation : pour chaque arc $(i,j) \in A$, et pour tout $k \in \tilde{K}$ tel que $x_{ij}^k > y_{ij}$, ajouter les inégalités valides correspondantes $x_{ij}^k \leq y_{ij}$ au PMR.

(f) Si les inégalités valides sont ajoutées au PMR :

i. Résoudre le nouveau PMR par l'algorithme dual du simplexe pour obtenir les solutions $\overline{y}, \overline{x}$ et la borne inférieure Z_{RL}.

ii. Si $Z_{RL} < Z^*$ et $\overline{y}$ entier, alors $Z^* = Z_{RL}$, $x^* = \overline{x}$ et $y^* = \overline{y}$.

iii. Si $Z_{RL} \geq Z^*$, aller à l'étape 5.

iv. Si $((Z_{RL} - Z_{Prec})/Z_{RL}) < \xi$, $++\varphi$, sinon $\varphi = 0$.
$Z_{Prec} = Z_{RL}$.

v. Si ($\varphi = max$) , alors aller à l'étape 5.

vi. Fixation des variables : Si $Z_{RL} + |\overline{f}_{ij}| \geq Z_{CD}$, alors $y_{ij} = \overline{y}_{ij}$, et aller à 3a.

vii. Aller à 3b.

4. Branchement : générer des nœuds enfants et les insérer dans Γ.

5. Si $\Gamma = \emptyset$, fin de l'algorithme. Autrement aller à l'étape 2.

Figure 4.1 – Algorithme de B&P&C.

Dans la première étape, une solution réalisable du problème MUND est calculée, elle identifie une borne supérieure sur la solution optimale du problème, et elle permet d'élaguer, le plus tôt possible, certains nœuds dans l'arbre de recherche. On définit aussi Γ l'ensemble des nœuds à évaluer qui est initialisé au nœud racine.

À l'étape 2, une règle de sélection est appliquée pour choisir le prochain nœud à évaluer, et le retirer de l'ensemble Γ.

Les différentes étapes de génération de colonnes et de génération de coupes appliquées à chaque nœud sont décrites dans la partie 3 de l'algorithme. En premier, et pour débuter la procédure de génération de colonnes, une solution réalisable est obtenue en résolvant par l'algorithme dual du simplexe le problème maître restreint, et dont nous assurons la réalisabilité par l'ajout des arcs artificiels. Ensuite, à l'étape 3b, une recherche

de nouvelles variables de flot x_{ij}^k qui peuvent améliorer la solution du problème maître (PM) est effectuée en calculant leurs coûts réduits. Si ces coûts sont négatifs (étape 3c), alors les variables de flot correspondantes sont ajoutées au problème PMR, lequel est résolu par l'algorithme primal du simplexe, avant de revenir à l'étape 3b. À la fin de la génération de colonnes, une borne inférieure Z_{RL} sur la valeur optimale du problème MUND est obtenue. Les tests d'élagage et d'arrêt sont introduits à l'étape 3d. Ces tests consistent à comparer Z_{RL} par rapport à la solution candidate Z^* premièrement, et deuxièmement par rapport à la solution précédente Z_{Prec}.

La génération de coupes débute à l'étape 3e, où l'on vérifie s'il existe des inégalités valides violées par la solution obtenue par la génération de colonnes. L'algorithme dual du simplexe est appliqué pour résoudre le nouveau problème maître si des nouvelles coupes sont ajoutées, suivi par des tests d'élagage et d'arrêt effectués sur la nouvelle borne inférieure (les étapes 3(f)i à 3(f)v). Ensuite, à l'étape 3(f)vi, la technique de fixation des variables est appliquée, et selon le test de fixation, la génération de colonnes reprend à nouveau à partir de l'une des étapes 3a ou 3b.

Après l'évaluation du nœud sélectionné, la règle de branchement (étape 4) sur les variables de conception y_{ij} est appliquée si nécessaire. Enfin, à l'étape 5, un test d'arrêt du processus est effectué pour vérifier si tous les nœuds sont explorés ou éliminés, autrement dit, si Γ est vide. Il est aussi possible de fixer une limite de temps après laquelle l'algorithme de B&P&C s'arrête.

CHAPITRE 5

RÉSULTATS

Nous présentons et interprétons dans ce chapitre les résultats expérimentaux de la méthode de Branch-and-Price-and-Cut (B&P&C) avec et sans critère d'arrêt, appliquée sur des instances de différentes tailles, en la comparant avec d'autres méthodes concurrentes comme la méthode de Branch-an-Bound (B&B) de Cplex.

5.1 Environnement

Les différentes expérimentations ont été réalisées sur un ordinateur Dell Studio muni de deux processeurs Intel(R) Xeon(R) X5675 cadencés à 3.07 GHz avec 96 Go de mémoire vive, le tout fonctionnant sous le système d'exploitation Linux.

L'algorithme proposé est implanté en C++, en utilisant la librairie logicielle Solving Contraint Integer Programs Scip (version 3.0.1). Scip est un solveur pour les programmes linéaires en nombres entiers (PLNE) et aussi pour les problèmes mixtes (MIP), développé dans le cadre de la thèse de doctorat de Tobias Achterberg [1]. Le solveur pour les relaxations linéaires n'est pas un composant interne de Scip, la raison pour laquelle il utilise des solveurs linéaires externes pour résoudre la relaxation du MIP. Dans ce mémoire, Cplex 12.5.1 est le solveur utilisé pour résoudre la relaxation linéaire de notre problème dans toutes les méthodes, que ce soit notre méthode de B&P&C ou bien la méthode de Branch-and-Cut (B&C).

5.2 Présentation des instances

Afin de préciser si l'algorithme de B&P&C avec et sans critère d'arrêt pour la conception de réseaux avec coûts fixes et sans capacité est suffisamment efficace et rapide, il est nécessaire de faire un certain nombre de tests sur différents types d'instances. Nous avons testé 156 instances provenant de deux sources principales. D'une part, 98 ins-

tances récupérées sur le site de A. Frangioni [1] qui sont largement utilisées par toutes les méthodes pour le problème avec capacités (voir l'article [21] pour la méthode de génération des instances). D'autre part, 58 instances obtenues du site du K. Holmberg[2]. Ces instances ont été utilisées dans deux articles [47, 48].
Pour obtenir nos instances, nous avons éliminé les capacités des instances citées.
Chaque instance est caractérisée par un triplet (N,A,P) qui définit sa taille et qui représente le nombre de sommets, d'arcs et de produits respectivement dans le réseau. Chacune de ces instances possède $|A|$ variables binaires et $|A||P|$ variables continues. Les instances sont divisées en 5 groupes représentés dans le tableau 5.I, en indiquant le nombre d'instances ayant la même taille par le nombre entre parenthèses.

5.3 Paramètres de performance

Dans cette section, on définit les termes que nous utilisons dans notre analyse :
- Temps : Le temps moyen d'exécution en secondes de chaque groupe d'instances pour chaque méthode.
- Variables : Le nombre moyen de variables de flot x_{ij}^k de chaque groupe d'instances générées par chaque méthode pour chaque groupe d'instances.
- Nœuds : Le nombre moyen de nœuds évalués par chaque algorithme de résolution.
- Gap : Représente l'écart relatif moyen par groupe d'instances entre la borne supérieure et la borne inférieure obtenue par chaque méthode. Lorsque le gap égal à 0, on retourne le nombre d'instances résolues à la racine.
- Gain de temps (%), Gain de variables (%) et Gain de nœuds (%) : Ce sont les gains relatifs moyens de temps, en nombre de variables et en nombre de nœuds, respectivement.

5.4 Analyse des résultats

Après avoir exécuté tous les algorithmes, et pour tester l'efficacité de notre méthode, nous avons varié d'abord les valeurs du critère d'arrêt, en affectant les valeurs 0,0001,

[1] `http://www.di.unipi.it/~frangio/`
[2] `http://http://www.mai.liu.se/~kajho48/problemdata/cnd/`

R-I	(28)	R-II	(27)	C	(31)	C+	(12)	H	(59)
(10,35,10)	(3)	(20,120,40)	(3)	(20,230,40)	(3)	(25,100,10)	(3)	([20-150],[255-814],< 16)	(13)
(10,35,25)	(3)	(20,120,100)	(3)	(20,230,200)	(4)	(25,100,30)	(3)	([30-100],[270-430],20)	(7)
(10,35,50)	(3)	(20,120,200)	(3)	(20,300,40)	(4)	(100,400,10)	(3)	([7-61],[42-930],30)	(4)
(10,50,50)	(1)	(20,220,40)	(3)	(20,300,200)	(4)	(100,400,30)	(3)	([8-61],[56-820],40)	(4)
(10,60,10)	(3)	(20,220,100)	(3)	(30,520,100)	(4)			([60-100],[400-510],50)	(3)
(10,60,25)	(3)	(20,220,200)	(3)	(30,520,400)	(4)			([10-40],[90-1000],80)	(3)
(10,60,50)	(3)	(20,314,40)	(3)	(30,690,100)	(4)			(10,[50-90],90)	(5)
(10,82,10)	(3)	(20,318,100)	(3)	(30,690,400)	(4)			([12-75],[100-1000],100,200[)	(8)
(10,83,25)	(3)	(20,315,200)	(3)					(20,[130-280],[200,300[)	(6)
(10,83,50)	(3)							(20,[180,310],[300,400[)	(4)
								([22,33],130, $\geq$ 400)	(2)

Tableau 5.I – Groupes d'instances.

0,0005, 0,001, 0,005, 0,01 et 0,05 au critère d'approximation ε, et de 2 à 7 pour le nombre d'itérations τ, afin d'introduire les meilleurs paramètres. Nous avons par la suite comparé ces résultats avec Cplex 12.5.1, l'un des meilleurs solveurs d'optimisation mathématique, en utilisant sa méthode de Branch-and-Bound (B&B), et en lui fournissant deux formulations fortes : une formulation à laquelle Cplex ajoute les inégalités fortes itérativement (dynamiquement), qu'on appelle la méthode dynamique, et une autre formulation où les inégalités fortes sont toutes introduites initialement dans le modèle qu'on appelle méthode statique. Notre méthode est comparée aussi avec la méthode de Branch-and-Cut, une version de notre méthode sans génération de colonnes développée avec Scip, et avec la méthode de Branch-and-Price-and-Cut sans intégrer le critère d'arrêt.
Cette comparaison est basée sur le temps moyen d'exécution et l'écart relatif pour les résultats à la racine, sur le temps moyen d'exécution pour les instances résolues pour lesquelles une solution optimale est obtenue dans un temps limité à trois heures (10800 secondes), sur l'écart relatif pour les instances qui ne sont pas résolues à l'optimum par toutes les méthodes dans le temps limité à trois heures, et finalement sur le nombre d'instances résolues pour les instances mixtes définies comme les instances résolues à l'optimum au moins par une, mais pas par toutes les méthodes de résolution présentées, dans un temps limité à trois heures.

En exploitant ces facteurs de comparaison dépendamment du type d'instance, on dit qu'une méthode est plus performante qu'une autre si elle résout les instances dans un temps plus court pour les instances résolues, ou si elle génère un écart relatif plus petit pour les instances non résolues, et enfin, si elle résout un plus grand nombre d'instances pour les instances mixtes.

Nous avons comparé ces méthodes d'abord à la racine, puis sur tout l'arbre de recherche en fixant une limite de temps à trois heures, et en utilisant comme bornes supérieures initiales les meilleures entre celles obtenues après avoir tourné toutes les méthodes pendant trois heures.

Nous identifions la méthode dynamique de Branch-and-Bound de Cplex par l'acronyme Cplex-D, la méthode statique de Branch-and-Bound de Cplex par l'acronyme Cplex-S, la méthode de Branch-and-Cut par l'acronyme BC, la méthode de Branch-and-Cut-and-Price par l'acronyme BPC, et finalement, la méthode de Branch-and-Cut-and-Price avec intégration de critère d'arrêt par l'acronyme BPC avec CA. Cette dernière est représentée dans les tableaux sous la forme par BPC-nombre d'itérations-Epsilon.

5.4.1 Comparaison à la racine

Nous avons appliqué toutes les méthodes à la racine en fixant la limite des nœuds évalués à un. À travers l'ensemble des tests qui ont été exécutés sur les instances de différentes tailles et qui sont rapportés dans le tableau 5.II, nous avons organisé notre comparaison selon deux principaux critères : l'écart relatif et le temps d'exécution.

Écart relatif : Comme le montre le tableau 5.II, l'écart relatif obtenu par la méthode de génération de coupes avec et sans génération de colonnes est presque le même écart obtenu par la méthode Cplex-S (1,28% versus 1,24%). Nous soulignons que l'écart relatif le plus élevé de toutes les méthodes est celui généré par la méthode dynamique de Cplex avec un gap de 4,05%, ce qui s'explique par le fait que Cplex n'intègre pas toutes les inégalités valides violées à la racine.

Nous remarquons aussi que l'écart relatif produit par la méthode de génération de coupes et de colonnes quand on intègre le processus d'arrêt est légèrement plus élevé par rapport à celles sans critère d'arrêt (1,50% versus 1,28%), à l'exception du cas où la valeur d'approximation (epsilon) est entre 0,005 et 0,05 et le nombre d'itérations est petit, l'écart relatif est alors plus élevé.

Temps d'exécution : On voit bien que notre méthode avec critère d'arrêt nécessite un temps moins élevé par rapport à la méthode BPC pour la résolution des instances de différentes tailles à la racine. En effet, un gain de temps jusqu'à 70% est obtenu avec une légère augmentation de l'écart relatif dans quelques cas. Un temps concurrentiel avec la méthode statique de Cplex est observé, notamment pour les instances difficiles (classe

Groupe	Méthode			Temps (s)	Variables	Gap (%)	Nbr-Ins-Rés
Moyenne	Cplex-D			0,35	34019	4,05	76
	Cplex-S			5,03	34019	1,24	99
	BC			12,31	34121	1,28	91
	BPC			10,50	5237	1,28	91
	BPC	2	0,0001	9,62	5227	1,55	89
			0,0005	8,79	5220	1,56	87
			0,001	8,38	5214	1,57	84
			0,005	6,27	5142	1,72	73
			0,01	4,92	5061	1,98	66
			0,05	2,50	4708	4,03	57
	BPC	3	0,0001	10,06	5232	1,50	91
			0,0005	9,33	5227	1,51	88
			0,001	8,79	5222	1,51	89
			0,005	7,66	5190	1,57	85
			0,01	6,14	5141	1,68	72
			0,05	3,41	4855	2,87	66
	BPC	4	0,0001	9,95	5232	1,50	91
			0,0005	9,92	5230	1,50	89
			0,001	9,49	5228	1,50	89
			0,005	7,73	5203	1,54	88
			0,01	7,40	5176	1,59	81
			0,05	4,03	4982	2,23	70
	BPC	5	0,0001	9,79	5232	1,50	91
			0,0005	9,99	5232	1,50	90
			0,001	9,66	5229	1,50	89
			0,005	8,65	5216	1,52	86
			0,01	7,58	5191	1,56	84
			0,05	4,63	5041	1,95	73
	BPC	6	0,0001	10,14	5232	1,50	91
			0,0005	10,49	5232	1,50	90
			0,001	8,99	5231	1,50	90
			0,005	8,69	5222	1,51	88
			0,01	8,30	5208	1,53	86
			0,05	5,47	5089	1,80	78
	BPC	7	0,0001	9,82	5232	1,50	91
			0,0005	10,11	5232	1,50	91
			0,001	9,97	5232	1,50	91
			0,005	8,82	5225	1,51	87
			0,01	8,44	5214	1,52	88
			0,05	6,12	5119	1,72	84

Tableau 5.II – Résultats à la racine.

C). Ces résultats nous encouragent à généraliser nos tests à tout l'arbre de recherche.

5.4.2 Comparaison des différentes méthodes sur tout l'arbre de recherche

Dans cette partie, nous analysons les résultats obtenus par toutes les méthodes en les divisant en groupes d'instances comme présentés dans le tableau 5.I. Sur l'ensemble des tests effectués sur les 156 instances, nous avons remarqué que l'utilisation de la génération de colonnes génère un gain en nombre de variables très important qui varie entre 69% et 86% par rapport aux autres méthodes. Nous avons aussi remarqué que les paramètres de critère d'arrêt qui génèrent de meilleurs résultats alternent généralement entre un nombre d'itérations $\tau = 2$ ou 3, avec une approximation $\varepsilon = 0,0001$ à $0,001$.

Groupe RI : Toutes les instances de ce groupe sont de petite taille (entre 10 et 50 produits), et elles sont toutes résolues par les différentes méthodes dans un temps très court qui ne dépasse pas une seconde, quelle que soit les valeurs des critères d'arrêt choisies, ou la technique d'exploration de l'arbre utilisée, ce qui rend difficile de conclure quelle méthode est meilleure. Les résultats sont rapportés dans le tableau 5.III.

Groupe	Méthode	Temps (s)	Nœuds	Variables
Groupe RI (28)	Cplex-D	0,02	2	1769
	Cplex-S	0,03	2	1769
	BC	0,07	2	1797
	BPC	0,04	2	520
	BPC-2-0,0005	0,05	2	520
	BPC-3-0,0005	0,05	2	521
	BPC-4-0,0005	0,05	2	520
	BPC-5-0,0005	0,05	2	521
	BPC-6-0,0005	0,04	2	520
	BPC-7-0,0005	0,04	2	520

Tableau 5.III – Groupe RI.

Groupe C+ : Ce groupe contient également des instances de petite taille. Le nombre de produits ne dépasse pas 30, et toutes les instances ont été résolues. Une lecture rapide

du tableau 5.IV permet de conclure que notre méthode (B&P&C avec CA) est concurrentielle avec Cplex, et elle est particulièrement meilleure que la méthode Cplex-D, avec un gain de temps de 50% quand on utilise la règle de sélection meilleur d'abord (Bfs).

Groupe H : Ce groupe contient 58 instances, et on y trouve 54 instances résolues, une instance mixte et 3 instances non résolues. Le tableau 5.V présente le résultat de ce groupe. Une lecture rapide du tableau nous permet de classifier les instances résolues comme instances faciles, puisque le temps moyen d'exécution est de moins d'une seconde pour presque toutes les méthodes. Il est de 2 s pour la méthode B&C et de 1,28 s pour le B&P&C, que ce soit avec la recherche meilleur d'abord (Bfs) ou bien la recherche en profondeur (Dfs).

Groupe RII : D'après les résultats présentés dans le tableau 5.VI, on dénombre 25 instances résolues, et deux instances mixtes. Pour la règle meilleur d'abord, le temps moyen d'exécution de la méthode de Cplex-D est de 180 s pour les instances résolues, 112 s pour la méthode Cplex-S, 421 s pour la méthode B&C, et 359 s pour la méthode B&P&C, tandis que le meilleur résultat pour la méthode B&P&C avec CA est de 226 s. Pour la règle en profondeur, le temps moyen d'exécution de la méthode Cplex-D est de 153 s pour les instances résolues, 151 s pour la méthode Cplex-S, 441 s pour la méthode B&C, et 339 s pour la méthode B&P&C, et le meilleur résultat pour la méthode B&P&C avec CA est de 203 s.
Pour les instances mixtes, les deux méthodes de Cplex résolvent les deux instances, la méthode B&P&C avec CA résout une instance sur deux, alors que les méthodes B&C et B&P&C n'en résolvent aucune.
Notons que la génération de colonnes génère seulement le quart des variables de flot sur les instances de l'ensemble RII.

Groupe C : Les résultats sont rapportés dans le tableau 5.VII. Sur l'ensemble de 31 instances, on dénombre 29 instances résolues, une instance mixte, et une instance non

Groupe	Méthode			Temps (s)	Nœuds	Variables
Groupe C+ (12)	Cplex-D			30,20	379	5251
	Cplex-S			10,15	30	5251
	BC			17,92	24	5270
	BPC			14,35	23	1613
	BPC	2	0,0001	11,10	29	1600
			0,0005	12,58	32	1605
			0,001	11,44	27	1598
			0,005	12,05	31	1604
			0,01	11,74	33	1610
			0,05	19,69	58	1633
	BPC	3	0,0001	10,67	23	1599
			0,0005	11,33	24	1600
			0,001	12,48	25	1603
			0,005	12,68	27	1595
			0,01	12,92	27	1605
			0,05	16,66	36	1628
	BPC	4	0,0001	12,19	23	1599
			0,0005	12,21	23	1601
			0,001	12,31	24	1607
			0,005	13,30	27	1602
			0,01	16,21	29	1613
			0,05	15,11	27	1621
	BPC	5	0,0001	14,22	26	1613
			0,0005	14,79	26	1610
			0,001	15,29	28	1612
			0,005	14,29	25	1600
			0,01	14,41	24	1606
			0,05	13,22	24	1609
	BPC	6	0,0001	14,21	24	1608
			0,0005	14,05	24	1608
			0,001	13,56	24	1612
			0,005	15,36	26	1615
			0,01	14,32	24	1607
			0,05	14,04	25	1610
	BPC	7	0,0001	13,10	24	1608
			0,0005	12,99	25	1610
			0,001	13,10	26	1615
			0,005	13,19	28	1601
			0,01	14,13	28	1614
			0,05	13,89	22	1608

(a) Instances résolues-Bfs

Groupe	Méthode			Temps (s)	Nœuds	Variables
Groupe C+ (12)	Cplex-D			11,20	347	5251
	Cplex-S			8,82	37	5251
	BC			18,14	25	5270
	BPC			15,09	29	1607
	BPC	2	0,0001	10,29	26	1594
			0,0005	11,85	29	1602
			0,001	12,46	30	1601
			0,005	11,37	30	1595
			0,01	11,06	30	1608
			0,05	14,87	44	1621
	BPC	3	0,0001	12,19	26	1608
			0,0005	11,93	26	1607
			0,001	13,58	30	1607
			0,005	16,67	41	1614
			0,01	13,88	31	1614
			0,05	13,10	33	1610
	BPC	4	0,0001	11,25	23	1601
			0,0005	11,44	24	1604
			0,001	12,86	26	1599
			0,005	14,56	28	1607
			0,01	13,74	31	1602
			0,05	13,72	27	1607
	BPC	5	0,0001	14,97	28	1616
			0,0005	14,83	28	1614
			0,001	14,76	30	1613
			0,005	13,09	25	1611
			0,01	16,45	28	1600
			0,05	16,49	32	1606
	BPC	6	0,0001	14,74	29	1610
			0,0005	15,80	29	1610
			0,001	15,06	28	1614
			0,005	15,23	30	1612
			0,01	14,88	28	1607
			0,05	14,09	27	1606
	BPC	7	0,0001	15,76	28	1611
			0,0005	14,81	28	1611
			0,001	14,34	29	1609
			0,005	14,72	29	1612
			0,01	15,00	29	1620
			0,05	12,80	24	1598

(b) Instances résolues-Dfs

Tableau 5.IV

Groupe C+.

Groupe	Méthode	Temps (s)	Nœuds	Variables
Groupe H	Cplex-D	0,33	6	25793
	Cplex-S	0,35	2	25793
	BC	1,96	2	25888
	BPC	1,28	3	3903
	BPC-2-0,0005	0,81	3	3886
	BPC-3-0,0005	0,83	2	3886
	BPC-4-0,0005	0,84	2	3886
	BPC-5-0,0005	0,83	2	3885
	BPC-6-0,0005	0,84	2	3886
	BPC-7-0,0005	0,83	2	3885

(a) Instances résolues-Bfs (54)

Groupe	Méthode	Temps (s)	Nœuds	Variables
Groupe H	Cplex-D	0,33	6	25793
	Cplex-S	0,35	2	25793
	BC	1,94	2	25888
	BPC	1,26	3	3903
	BPC-2-0,0005	0,81	3	3886
	BPC-3-0,0005	0,82	2	3886
	BPC-4-0,0005	0,83	2	3885
	BPC-5-0,0005	0,84	2	3885
	BPC-6-0,0005	0,82	2	3885
	BPC-7-0,0005	0,82	2	3885

(b) Instances résolues-Dfs (54)

Groupe	Méthode	Gap (%)	Nœuds	Variables
Groupe H	Cplex-D	2,60	7426	61374
	Cplex-S	3,00	2700	61374
	BC	4,57	535	61613
	BPC	4,75	434	24590
	BPC-2-0,0005	4,21	706	24798
	BPC-3-0,0005	4,37	597	24831
	BPC-4-0,0005	4,53	542	24688
	BPC-5-0,0005	4,63	512	24540
	BPC-6-0,0005	4,65	491	24508
	BPC-7-0,0005	4,63	468	24513

(c) Instances non résolues (3)

Groupe	Parcours	Méthode	Nb inst	Parcours	Méthode	Nb inst
Groupe H	Bfs	Cplex-D	1	Dfs	Cplex-D	1
		Cplex-S	1		Cplex-S	1
		BC	0		BC	1
		BPC	1		BPC	1
		BPC-2-0,0005	1		BPC-2-0,0005	1
		BPC-3-0,0005	1		BPC-3-0,0005	1
		BPC-4-0,0005	1		BPC-4-0,0005	1
		BPC-5-0,0005	1		BPC-5-0,0005	1
		BPC-6-0,0005	1		BPC-6-0,0005	1
		BPC-7-0,0005	1		BPC-7-0,0005	1

(d) Instances mixtes (1)

Tableau 5.V – Groupe H.

résolue. Lorsque la règle de parcours meilleur d'abord est appliquée, le meilleur temps d'exécution par notre méthode est de 194 s, 287 s pour la méthode Cplex-S, 355 s pour la méthode B&C, 338 s pour la méthode B&P&C, et dernièrement 170 s par la méthode Cplex-D.

Pour le parcours en profondeur, on remarque que la résolution des instances nécessite un peu plus de temps pour certaines méthodes. Notre méthode nécessite alors 228 s dans le meilleur cas, tandis que la méthode B&C prend 378 s. On observe le même temps d'exécution pour la méthode Cplex-D (170 s), et un temps meilleur que le parcours meilleur d'abord pour les méthodes Cplex-S avec 186 s, et B&P&C avec 303 s.

Pour l'instance non résolue, nous observons que l'écart relatif généré par notre méthode est égal à 1,73%, versus des écarts de 2,10%, 2,02%, 1.91% et 1,10% générés par Cplex-S, B&P&C, B&C et Cplex-D respectivement.

Pour l'instance mixte, on remarque que la méthode B&C n'arrive pas à la résoudre dans le temps limité à trois heures quel que soit le parcours appliqué. La méthode Cplex-S avec Dfs n'est pas capable de la résoudre non plus. Pour les méthodes qui l'ont résolue, le meilleur temps constaté est celui de notre méthode B&P&C avec CA avec 4205 s, sui-

Groupe	Méthode			Temps (s)	Nœuds	Variables
Groupe RII	Cplex-D			180,93	1246	22691
	Cplex-S			112,23	114	22691
	BC			421,97	101	22796
	BPC			359,03	112	5905
	BPC	2	0,0001	242,16	163	5834
			0,0005	226,37	163	5840
			0,001	250,17	169	5851
			0,005	262,65	192	5844
			0,01	328,38	246	5917
			0,05	764,36	556	6172
	BPC	3	0,0001	246,35	135	5852
			0,0005	257,84	133	5863
			0,001	239,51	133	5841
			0,005	259,58	148	5850
			0,01	247,34	142	5846
			0,05	321,30	170	5923
	BPC	4	0,0001	262,84	127	5862
			0,0005	257,12	126	5858
			0,001	269,57	128	5868
			0,005	271,63	122	5870
			0,01	277,59	127	5865
			0,05	350,52	167	5920
	BPC	5	0,0001	267,50	118	5858
			0,0005	270,20	119	5859
			0,001	267,20	117	5854
			0,005	249,55	120	5867
			0,01	283,02	117	5864
			0,05	283,35	121	5886
	BPC	6	0,0001	275,19	113	5872
			0,0005	278,27	116	5878
			0,001	271,56	115	5866
			0,005	265,65	113	5856
			0,01	267,23	110	5883
			0,05	297,61	114	5886
	BPC	7	0,0001	286,50	110	5879
			0,0005	269,45	109	5877
			0,001	262,62	109	5874
			0,005	263,20	112	5881
			0,01	278,40	112	5876
			0,05	304,57	108	5891

(a) Instances résolues-Bfs (25)

Groupe	Méthode			Temps (s)	Nœuds	Variables
Groupe RII	Cplex-D			153,54	1770	22691
	Cplex-S			151,88	133	22691
	BC			441,32	103	22796
	BPC			339,25	106	5902
	BPC	2	0,0001	243,31	161	5838
			0,0005	203,04	151	5815
			0,001	230,52	161	5829
			0,005	252,86	186	5839
			0,01	339,30	247	5913
			0,05	708,63	531	6142
	BPC	3	0,0001	242,77	134	5842
			0,0005	237,48	131	5836
			0,001	229,72	132	5826
			0,005	243,85	142	5844
			0,01	227,98	131	5814
			0,05	321,38	174	5925
	BPC	4	0,0001	243,08	119	5848
			0,0005	243,72	122	5849
			0,001	263,11	131	5863
			0,005	255,83	121	5849
			0,01	277,60	134	5853
			0,05	290,91	139	5902
	BPC	5	0,0001	266,58	116	5873
			0,0005	250,41	114	5856
			0,001	252,07	116	5858
			0,005	232,59	116	5838
			0,01	278,28	113	5862
			0,05	256,49	119	5870
	BPC	6	0,0001	270,28	117	5862
			0,0005	285,43	115	5867
			0,001	278,45	115	5868
			0,005	277,13	119	5851
			0,01	258,22	120	5871
			0,05	270,22	109	5853
	BPC	7	0,0001	281,61	117	5867
			0,0005	267,10	116	5872
			0,001	269,10	117	5854
			0,005	251,52	114	5847
			0,01	277,82	119	5870
			0,05	304,59	118	5876

(b) Instances résolues-Dfs (25)

Groupe	Parcours	Méthode	Nb inst	Parcours	Méthode	Nb inst
Groupe RII	Bfs	Cplex-D	2	Dfs	Cplex-D	2
		Cplex-S	2		Cpex-S	2
		BC	0		BC	0
		BPC	0		BPC	0
		BPC-2-0,0005	0		BPC-2-0,0005	0
		BPC-3-0,0005	1		BPC-3-0,0005	1
		BPC-4-0,0005	1		BPC-4-0,0005	1
		BPC-5-0,0005	1		BPC-5-0,0005	1
		BPC-6-0,0005	1		BPC-6-0,0005	0
		BPC-7-0,0005	1		BPC-7-0,0005	0

(c) Instances mixtes (2)

Tableau 5.VI – Groupe RII.

vie par la méthode Cplex-D avec un temps de 4588 s.

Ajoutons aussi que notre méthode ne génère que 14% du nombre de variables générées par les méthodes sans génération de colonnes.

Instances de grande taille

Pour apprécier réellement notre méthode, nous avons classifié l'ensemble des instances testées par les différentes méthodes dans ce travail, selon le nombre de produits qu'elles contiennent. Nous distinguons 14 instances de grande taille ayant au moins 300 produits provenant des groupes RII, C et H. On dénombre 12 instances résolues, une instance non résolue et une instance mixte.

Pour ce groupe d'instances, la méthode B&P&C sans et avec critère d'arrêt performe mieux que la méthode dynamique et statique de Cplex, lorsqu'on applique la règle de sélection en profondeur.

En effet, une comparaison de BPC avec Cplex-D, nous permet d'obtenir un gain de 38,51% en temps, 97,97 % en nombre de nœuds et 88,41% en nombre de variables. Par rapport à la méthode Cplex-S, BPC génère des gains de 28,14%, 12,46% et 88,41% en temps, en nombre de nœuds et en nombre de variables respectivement. Pour la méthode

Groupe	Méthode			Temps (s)	Nœuds	Variables
Groupe C	Cplex-D			170,02	1587	81310
	Cplex-S			287,36	303	81310
	BC			355,25	180	81485
	BPC			338,01	175	10973
	BPC	2	0,0001	288,93	240	10919
			0,0005	253,05	194	10905
			0,001	251,39	191	10907
			0,005	318,56	257	10970
			0,01	374,86	306	11075
			0,05	1239,81	865	11989
	BPC	3	0,0001	285,00	211	10925
			0,0005	288,91	208	10921
			0,001	258,17	190	10909
			0,005	265,81	175	10909
			0,01	194,95	181	10926
			0,05	415,71	240	11299
	BPC	4	0,0001	280,32	192	10913
			0,0005	286,74	203	10931
			0,001	296,77	205	10922
			0,005	271,78	176	10928
			0,01	269,02	173	10929
			0,05	319,46	196	11043
	BPC	5	0,0001	297,52	186	10939
			0,0005	303,52	195	10945
			0,001	302,05	195	10942
			0,005	279,40	168	10931
			0,01	271,98	166	10927
			0,05	307,99	190	10973
	BPC	6	0,0001	298,46	186	10941
			0,0005	291,84	178	10930
			0,001	295,78	183	10939
			0,005	274,92	156	10944
			0,01	276,26	156	10931
			0,05	297,13	181	10942
	BPC	7	0,0001	296,47	179	10915
			0,0005	286,26	178	10958
			0,001	300,95	201	10967
			0,005	264,59	167	10919
			0,01	267,92	170	10928
			0,05	281,04	162	10934

(a) Instances résolues-Bfs (29)

Groupe	Méthode			Temps (s)	Nœuds	Variables
Groupe C	Cplex-D			170,57	2306	81310
	Cplex-S			186,42	274	81310
	BC			378,15	211	81485
	BPC			303,99	172	10924
	BPC	2	0,0001	283,24	265	10903
			0,0005	250,52	225	10905
			0,001	228,58	184	10879
			0,005	290,72	239	10944
			0,01	324,63	280	11037
			0,05	1165,15	898	11917
	BPC	3	0,0001	273,92	223	10901
			0,0005	266,49	220	10891
			0,001	243,87	185	10877
			0,005	234,50	172	10890
			0,01	248,96	174	10892
			0,05	387,82	239	11261
	BPC	4	0,0001	253,76	183	10898
			0,0005	256,19	181	10910
			0,001	273,34	204	10908
			0,005	256,15	182	10899
			0,01	255,49	182	10897
			0,05	297,66	187	10978
	BPC	5	0,0001	280,00	189	10908
			0,0005	275,03	190	10902
			0,001	276,49	192	10897
			0,005	260,66	180	10897
			0,01	264,32	182	10898
			0,05	301,40	197	10941
	BPC	6	0,0001	303,00	203	10921
			0,0005	288,62	200	10909
			0,001	278,67	181	10910
			0,005	269,27	180	10894
			0,01	270,74	182	10908
			0,05	289,71	172	10901
	BPC	7	0,0001	305,23	210	10911
			0,0005	277,58	210	10917
			0,001	278,92	205	10902
			0,005	258,29	181	10897
			0,01	256,42	180	10901
			0,05	276,22	180	10889

(b) Instances résolues-Dfs (29)

Groupe	Méthode	Gap (%)	Nœuds	Variables
Groupe C	Cplex-D	1,10	14706	272280
	Cplex-S	2,10	273	272280
	BC	1,91	321	272679
	BPC	2,02	269	38191
	BPC-2-0,0005	1,90	734	38931
	BPC-3-0,0005	1,83	585	38865
	BPC-4-0,0005	1,79	522	38803
	BPC-5-0,0005	1,80	490	38839
	BPC-6-0,0005	1,75	452	38551
	BPC-7-0,0005	1,79	477	38715

(c) Instances non résolues (1)

Groupe	Parcours	Méthode	Nb inst	Parcours	Méthode	Nb inst
Groupe C	Bfs	Cplex-D	1	Dfs	Cplex-D	1
		Cplex-S	0		Cplex-S	0
		BC	0		BC	0
		BPC	1		BPC	1
		BPC-2-0,0005	0		BPC-2-0,0005	0
		BPC-32-0,0005	1		BPC-3-0,0005	1
		BPC-4-0,0005	1		BPC-4-0,0005	1
		BPC-5-0,0005	1		BPC-5-0,0005	1
		BPC-6-0,0005	1		BPC-6-0,0005	1
		BPC-7-0,0005	1		BPC-7-0,0005	1

(d) Instances mixtes (1)

Tableau 5.VII – Groupe C.

B&P&C avec CA, les gains atteignent jusqu'à 52% en temps, 97% en nombre de nœuds et 85% en nombre de variables par rapport à la méthode de Cplex-D. Tandis que versus la méthode Cplex-S, la méthode BPC avec CA génère 44% de gain en temps, 85% en nombre de variables. Par contre, le nombre de nœuds évalués par BPC avec CA est beaucoup plus élevé par rapport à Cplex-S.

Pour la règle meilleur d'abord, la méthode BPC est meilleure que Cplex-S (83 s versus 108 s). De plus, elle est compétitive avec Cplex-D (82 s). Alors que la méthode BPC avec CA est meilleure que les deux méthodes de Cplex, avec des gains du temps de 25% par rapport à Cplex-D, et 43% par rapport à Cplex-S. Quelle que soit la méthode de Cplex, le gain obtenu en nombre de variables atteint 85%, ainsi qu'un gain jusqu'à 92% en nombre de nœuds obtenu seulement par rapport à Cplex-D.

Finalement, relativement à la méthode B&C, la méthode B&P&C sans et avec critère d'arrêt est toujours meilleure non seulement pour les instances de grande taille, mais également pour les instances de petite et moyenne tailles.

Pour les instances non résolue et mixte, notre méthode reste compétitive avec toutes les autres méthodes en raison de leurs résultats très proches (voir tableau 5.VIII).

5.4.3 L'impact du critère d'arrêt dans la méthode de Branch-and-Price-and-Cut

Nous remarquons que l'intégration des paramètres d'arrêts (nombre d'itérations et valeur d'approximation) provoque une perturbation par rapport à BPC, au niveau du temps d'exécution, et du nombre de nœuds évalués. En effet, l'intégration de ce type de critère affecte positivement nos résultats pour les différents types d'instances. Un gain moyen de temps de 20% est ainsi obtenu par itération pour presque toutes les instances. Nous remarquons aussi qu'il y a une relation proportionnelle entre la taille des instances et le nombre de nœuds évalués. Lorsque le nombre de produits dans une instance est grand, le nombre de nœuds générés augmente par rapport à la méthode de B&P&C. Pour les instances de grande taille (300 produits et plus), le double du nombre de nœuds est évalué par B&P&C avec CA par rapport à BPC, et ce dans un temps meilleur.

Finalement, nous soulignons que, quel que soit le critère d'arrêt appliqué, les deux méthodes génèrent approximativement le même nombre de variables..

Groupe	Méthode			Temps (s)	Nœuds	Variables
Instances de grande taille	Cplex-D			119,76	1123	150150
	Cplex-S			102,48	26	150150
	BC			139,88	23	150529
	BPC			73,64	23	17405
	BPC	2	0,0001	64,90	58	21103
			0,0005	61,07	54	21109
			0,001	63,15	56	21086
			0,005	80,29	67	21176
			0,01	91,16	85	21327
			0,05	638,10	559	23535
	BPC	3	0,0001	67,50	54	21095
			0,0005	62,93	45	21085
			0,001	62,69	44	21047
			0,005	63,71	44	21083
			0,01	65,62	50	21103
			0,05	160,48	96	22036
	BPC	4	0,0001	68,69	45	21095
			0,0005	72,32	53	21110
			0,001	69,85	51	21109
			0,005	63,80	39	21084
			0,01	63,74	43	21075
			0,05	76,31	46	21243
	BPC	5	0,0001	65,44	48	21095
			0,0005	70,58	42	21077
			0,001	73,44	45	21110
			0,005	62,68	36	21053
			0,01	65,16	42	21059
			0,05	70,70	42	21157
	BPC	6	0,0001	70,70	39	21111
			0,0005	64,24	39	21089
			0,001	68,05	44	21115
			0,005	60,79	34	21085
			0,01	63,63	35	21060
			0,05	72,11	38	21086
	BPC	7	0,0001	67,74	36	21098
			0,0005	62,90	36	21103
			0,001	67,11	44	21106
			0,005	63,60	35	21069
			0,01	56,99	32	21072
			0,05	65,58	34	21084

(a) Instances résolues-Dfs (12)

Groupe	Méthode			Temps (s)	Nœuds	Variables
Instances de grande taille	Cplex-D			82,74	328	150150
	Cplex-S			108,48	32	150150
	BC			132,41	19	150529
	BPC			83,27	24	17432
	BPC	2	0,0001	71,41	61	21117
			0,0005	63,81	53	21096
			0,001	65,29	58	21098
			0,005	86,83	82	21233
			0,01	104,33	94	21351
			0,05	722,99	610	23687
	BPC	3	0,0001	69,73	57	21130
			0,0005	63,25	47	21098
			0,001	61,28	43	21058
			0,005	66,40	50	21078
			0,01	72,60	53	21145
			0,05	170,98	98	21885
	BPC	4	0,0001	65,39	42	21079
			0,0005	64,25	41	21140
			0,001	66,30	42	21065
			0,005	65,19	39	21087
			0,01	69,36	41	21127
			0,05	84,24	55	21320
	BPC	5	0,0001	70,20	42	21133
			0,0005	69,76	43	21102
			0,001	71,02	43	21120
			0,005	65,55	40	21115
			0,01	67,31	42	21100
			0,05	69,76	36	21168
	BPC	6	0,0001	72,61	44	21108
			0,0005	71,17	43	21117
			0,001	72,97	44	21144
			0,005	61,84	37	21104
			0,01	64,93	33	21092
			0,05	68,38	44	21102
	BPC	7	0,0001	69,29	46	21113
			0,0005	73,10	51	21157
			0,001	69,45	47	21142
			0,005	66,38	42	21112
			0,01	61,79	36	21108
			0,05	65,81	37	21096

(b) Instances résolues-Bfs (12)

Groupe	Méthode	Gap (%)	Nœuds	Variables
Instances de grande taille	Cplex-D	1,10	14706	272280
	Cplex-S	2,10	273	272280
	BC	1,91	321	272679
	BPC	2,02	269	38191
	BPC-2-0,0005	1,90	734	38931
	BPC-3-0,0005	1,83	585	38865
	BPC-4-0,0005	1,79	522	38803
	BPC-5-0,0005	1,80	490	38839
	BPC-6-0,0005	1,75	452	38551
	BPC-7-0,0005	1,79	477	38715

(c) Instances non résolues (1)

Groupe	Parcours	Méthode	Nb.Inst	Parcours	Méthode	Nb.Inst
Instances de grande taille	Bfs	Cplex-D	1	Dfs	Cplex-D	1
		Cplex-S	1		Cplex-S	0
		BC	0		BC	0
		BPC	1		BPC	1
		BPC-2-0,0005	0		BPC-2-0,0005	0
		BPC-3-0,0005	1		BPC-3-0,0005	1
		BPC-4-0,0005	1		BPC-4-0,0005	1
		BPC-5-0,0005	1		BPC-5-0,0005	1
		BPC-6-0,0005	1		BPC-6-0,0005	1
		BPC-7-0,0005	1		BPC-7-0,0005	1

(d) Instances mixtes (1)

Tableau 5.VIII – Instances de grande taille.

Dans les Figures 5.1 et 5.2, nous illustrons l'impact du critère d'arrêt sur la méthode de B&P&C pour les instances de moyenne et grande tailles (100 produits et plus), en les distinguant selon le nombre de produits. Nous y représentons ainsi le temps moyen d'exécution et le nombre moyen de nœuds explorés en fonction de la valeur d'approximation ε pour chaque limite du nombre d'itérations τ.

Figure 5.1 – Instances de grande taille : Temps.

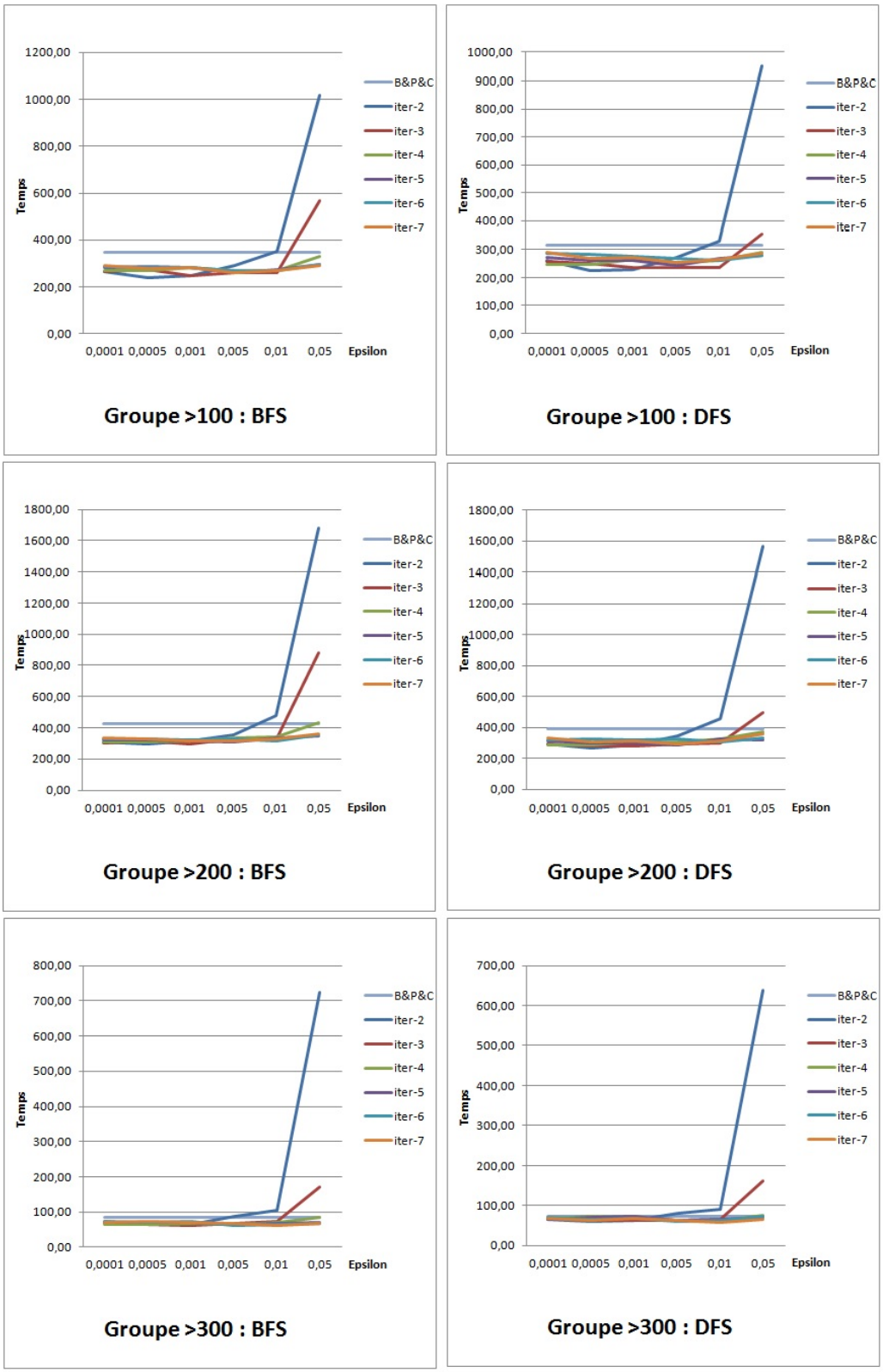
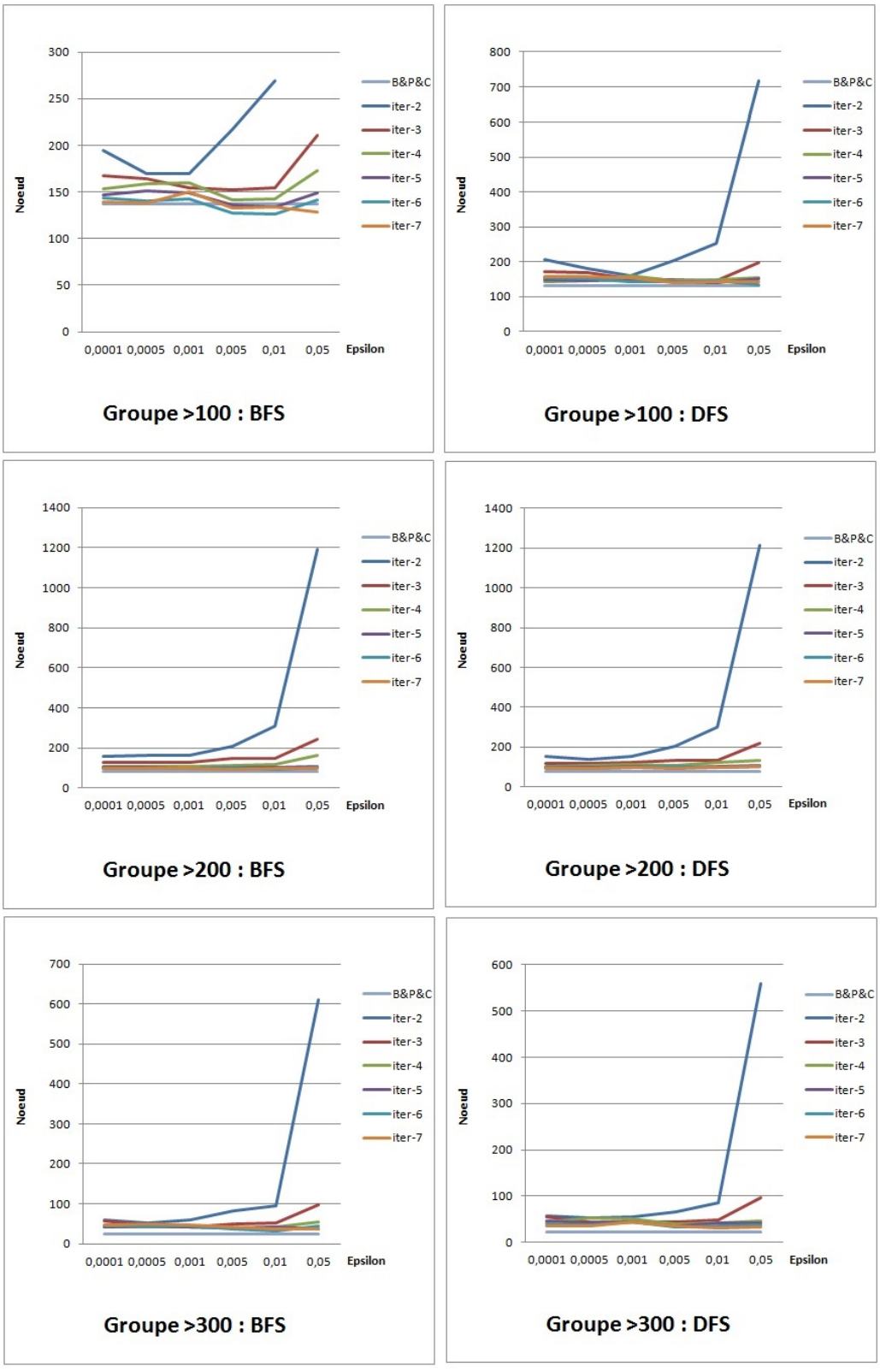


Figure 5.2 – Instances de grande taille : Nœuds.



CHAPITRE 6

CONCLUSION

Nous avons présenté dans ce mémoire une méthode exacte basée sur des techniques de programmation linéaire en nombres entiers pour résoudre le problème de conception de réseaux avec coûts fixes et sans capacité.

Nous avons appliqué la méthode de Branch-and-Bound basée sur la relaxation linéaire tout en exploitant les méthodes de génération de colonnes et de génération de coupes. Dans le but d'accélérer la méthode, nous avons utilisé un critère d'arrêt basé sur un facteur de convergence permettant de suspendre le processus de résolution lorsque la solution ne s'améliore pas suffisamment après un certain nombre d'itérations.

En se comparant à Cplex, un des meilleurs logiciels d'optimisation mathématique, ainsi qu'à une méthode de Branch-and-Cut, notre méthode s'est montrée compétitive et rapide, en particulier, pour les instances de taille moyenne et supérieure ayant un grand nombre de produits.

Certains développements futurs peuvent maintenant être envisagés pour l'amélioration de notre méthode, il serait intéressant d'incorporer de nouvelles inégalités valides, dans le but de diminuer le nombre de nœuds générés dans l'arbre de recherche, et par conséquent, réduire le temps d'exécution. Nous proposons aussi d'utiliser une heuristique qui remplace la méthode exacte de Branch-and-Bound afin de limiter le nombre de noeuds visités.

Finalement, bien que les instances sur lesquelles nous avons effectué notre expérimentation contenaient jusqu'à 460 produits, nous suggérons d'appliquer notre méthode sur des problèmes ayant un plus grand nombre de produits pour mieux en mesurer la qualité des résultats.

BIBLIOGRAPHIE

[1] T. Achterberg. Scip : solving constraint integer programs. *Mathematical Programming Computation*, 1:1–41, 2009.

[2] T. Achterberg, T. Koch et A. Martin. Branching rules revisited. *Operations Research Letters*, 33:42–54, 2005.

[3] A. Atamtürk. On capacitated network design cut-set polyhedra. *Mathematical Programming*, 92:425–437, 2000.

[4] A. Atamtürk et O. Günlük. Network design arc set with variable upper bounds. *Networks*, 50:17–28, 2007.

[5] A. Atamtürk et D. Rajan. On splittable and unsplittable flow capacitated network design arc-set polyhedra. *Mathematical Programming*, 92:315–333, 2002.

[6] A. Balakrishnan, T. L. Magnanti et R. T. Wong. A dual-ascent procedure for large-scale uncapacitated network design. *Operations Research*, 37:716–740, 1989.

[7] C. Barnhart. Dual-ascent methods for large-scale multicommodity flow problems. *Naval Research Logistics*, 40:305–324, 1993.

[8] J.F. Benders. Partitioning procedures for solving mixed-variables programming problems. *Numerische Mathematik*, 4:238–252, 1962.

[9] D. Bienstock et O. Günlük. Capacitated network design – polyhedral structure and computation. *Informs Journal on Computing*, 8:243–259, 1994.

[10] J. W. Billheimer et P. Gray. Network design with fixed and variable cost elements. *Transportation Science*, 7:49–74, 1973.

[11] T.B. Boffey et A.I. Hinxman. Solving the optimal network problem. *European Journal of Operational Research*, 3:386 – 393, 1979.

[12] D.E. Boyce, A . Farhi et R. Weischedel. Optimal network problem : a branch-and-bound algorithm. *Environment and Planning*, 5:519–533, 1973.

[13] M. Chouman et T. G. Crainic. A mip-tabu search hybrid framework for multicommodity capacitated fixed-charge network design. Rapport technique CIRRELT-2010-31, Centre interuniversitaire de recherche sur les réseaux d'entreprise, la logistique et le transport (CIRRELT), 2010.

[14] M. Chouman, T. G. Crainic et B. Gendron. A cutting-plane algorithm based on cutset inequalities for multicommodity capacitated fixed charge network design. Rapport technique CIRRELT-2009-20, Centre interuniversitaire de recherche sur les réseaux d'entreprise, la logistique et le transport (CIRRELT), 2009.

[15] M. Chouman, T. G. Crainic et B. Gendron. The impact of filtering in a branch-and-cut algorithm for multicommodity capacitated fixed charge network design. Rapport technique CIRRELT-2014-35, Centre interuniversitaire de recherche sur les réseaux d'entreprise, la logistique et le transport (CIRRELT), 2014.

[16] M. Chouman, T.G. Crainic et B. Gendron. Revue des inégalités valides pertinentes aux problèmes de conception de réseaux. *INFOR*, 41:5–33, 2003.

[17] A. M. Costa, J. Cordeau et B. Gendron. Benders, metric and cutset inequalities for multicommodity capacitated network design. *Computational Optimization and Applications*, 42:371–392, 2009.

[18] T. G. Crainic. Service network design in freight transportation. *European Journal of Operational Research*, 122:272–288, 2000.

[19] T. G. Crainic, M. Gendreau et J. M. Farvolden. A simplex-based tabu search method for capacitated network design. *INFORMS Journal on Computing*, 12:223–236, 2000.

[20] T. G. Crainic, Y. Li et M. Toulouse. A first multilevel cooperative algorithm for capacitated multicommodity network design. *Computers & Operations Research*, 33:2602–2622, 2006.

[21] T.G. Crainic, A. Frangioni et B. Gendron. Bundle-based relaxation methods for multicommodity capacitated fixed charge network design. *Discrete Applied Mathematics*, 112:73–99, 2001.

[22] T.G. Crainic et M. Gendreau. Cooperative parallel tabu search for capacitated network design. *Journal of Heuristics*, 8:601–627, 2002.

[23] M. Dell'Amico, F. Maffioli et S. Martello. *Annotated bibliographies in combinatorial optimization*. Wiley-Interscience series in discrete mathematics and optimization. Wiley, 1997.

[24] E.W. Dijkstra. A note on two problems in connexion with graphs. *Numerische Mathematik*, 1:269–271, 1959.

[25] R. Dionne et M. Florian. Exact and approximate algorithms for optimal network design. *Networks*, 9:37–59, 1979.

[26] Y. Dodge. *Optimisation appliquée*. Statistique et probabilités Appliquées Series. Springer, 2004.

[27] S. El Filali. Méthode de génération de colonnes pour les problèmes de conception de réseaux avec coûts d'ajout de capacité. Mémoire de maîtrise, Université de Montréal, 2014.

[28] D. Erlenkotter. A dual-based procedure for uncapacitated facility location. *Operations Research*, 26:992–1009, 1978.

[29] M. Fischetti et A. Lodi. Local branching. *Mathematical Programming*, 98:23–47, 2003.

[30] M. Fischetti et P. Toth. An additive bounding procedure for combinatorial optimization problems. *Operations Research*, 37:319–328, 1989.

[31] A. Frangioni et B. Gendron. 0–1 reformulations of the multicommodity capacitated network design problem. *Discrete Applied Mathematics*, 157:1229–1241, 2009.

[32] G. Gallo. A new branch and bound algorithm for the network design problem. Rapport technique L81-01, Istituto Di Elaborazione Della informazione, 1981.

[33] B. Gendron. A note on : a dual-ascent approach to the fixed-charge capacitated network design problem. *European Journal of Operational Research*, 138:671–675, 2002.

[34] B. Gendron. Decomposition methods for network design. *Procedia - Social and Behavioral Sciences*, 20:31–37, 2011.

[35] B. Gendron et T. G. Crainic. Relaxations for multicommodity capacitated network design problems. Rapport technique CRT-965, Centre for Research on Transportation (CRT), 1994.

[36] B. Gendron et T. G. Crainic. A branch-and-bound algorithm for depot location and container fleet management. *Location Science*, 3:39–53, 1995.

[37] B. Gendron et T. G. Crainic. Bounding procedures for multicommodity capacitated fixed charge network design problems. Rapport technique CRT-96-06, Centre for Research on Transportation (CRT), 1996.

[38] B. Gendron, T.G. Crainic et A. Frangioni. Multicommodity capacitated network design. Dans *Telecommunications Network Planning*, Centre for Research on Transportation (CRT), pages 1–19. Springer US, 1999.

[39] B. Gendron et M. Larose. Branch-and-price-and-cut for large-scale multicommodity capacitated fixed-charge network design. *EURO Journal on Computational Optimization*, 95:1–21, 2014.

[40] I. Ghamlouche, T. G. Crainic et M. Gendreau. Cycle-based neighbourhoods for fixed-charge capacitated multicommodity network design. *Operations Research*, 51:655–667, 2003.

[41] I. Ghamlouche, T.G. Crainic et M. Gendreau. Path relinking, cycle-based neighbourhoods and capacitated multicommodity network design. *Annals of Operations Research*, 131:109–133, 2004.

[42] J. Hellstrand, T. Larsson et A. Migdalas. A characterization of the uncapacitated network design polytope. *Operations Research Letters*, 12:159–163, 1992.

[43] G. Hernu. Heuristiques basées sur la programmation mathématique pour le problème de conception de réseaux avec coûts fixes et capacités. Mémoire de maîtrise, Université de Montréal, 2001.

[44] J.W. Herrmann, G. Ioannou, I. Minis et J.M. Proth. A dual ascent approach to the fixed-charge capacitated network design problem. *European Journal of Operational Research*, 95:476–490, 1996.

[45] H. H. Hoang. A computational approach to the selection of an optimal network. *Management Science*, 19:488–498, 1973.

[46] K. Holmberg et J. Hellstrand. Solving the uncapacitated network design problem by a lagrangean heuristic and branch-and-bound. *Operations Research*, 46:247–259, 1998.

[47] K. Holmberg et D Yuan. A lagrangean approach to network design problems. *International Transactions in Operational Research*, 5:529–539, 1998.

[48] K. Holmberg et D. Yuan. A lagrangian heuristic based branch-and-bound approach for the capacitated network design problem. *Operations Research*, 48:461–481, 2000.

[49] M. Junger, T.M. Liebling, D. Naddef, G.L. Nemhauser, W.R. Pulleyblank, G. Reinelt, G. Rinaldi et L.A. Wolsey. *50 Years of Integer Programming 1958-2008*. Springer Berlin Heidelberg, 2009.

[50] N. Katayama, M. Chen et M. Kubo. A capacity scaling heuristic for the multi-commodity capacitated network design problem. *Journal of Computational and Applied Mathematics*, 232:90 – 101, 2009.

[51] D. Kim et C. Barnhart. *Transportation service network design : Models and algorithms*. Springer, 1999.

[52] D. Kim, C. Barnhart, K. Ware et G. Reinhardt. Multimodal express package delivery : A service network design application. *Transportation Science*, 33:391–407, 1999.

[53] G. Kliewer et L. Timajev. Relax-and-cut for capacitated network design. Dans GerthStolting Brodal et Stefano Leonardi, éditeurs, *Algorithms ESA 2005*, volume 3669 de *Lecture Notes in Computer Science*, pages 47–58. Springer Berlin Heidelberg, 2005.

[54] J. Krarup et P. M. Pruzan. The simple plant location problem : Survey and synthesis. *European Journal of Operational Research*, 12:36–81, 1983.

[55] J. Kratica, D. Tosic, V. Filipovic et I. Ljubic. A genetic algorithm for the uncapacitated network design problem. Dans Rajkumar Roy, Mario Kï£¡ppen, Seppo Ovaska, Takeshi Furuhashi et Frank Hoffmann, éditeurs, *Soft Computing and Industry*, pages 329–336. Springer London, 2002.

[56] V. Lakshmi. Genetic algorithm for uncapacitated network design. Mémoire de maîtrise, Massachusettes Institue of Technology, 1995.

[57] B. W. Lamar, Y. Sheffi et W. B. Powell. *Bounding Procedures for Fixed Charge, Multicommodity Network Design Problems.*, 1984.

[58] M. Larose. Développement d'un algorithme de branch-and-price-and-cut pour le problème de conception de réseau avec coûts fixes et capacités. Mémoire de maîtrise, Université de Montréal, 2011.

[59] J. T. Linderoth et M. W. P. Savelsbergh. A computational study of search strategies for mixed integer programming. *INFORMS*, 11:173–187, 1999.

[60] M. Los et C. Lardinois. Combinatorial programming, statistical optimization and the optimal transportation network problem. *Transportation Research Part B : Methodological*, 16:89–124, 1982.

[61] T. L Magnanti et R. T. Wong. Network design and transportation planning : Models and algorithms. *Transportation Science*, 18:1–55, 1984.

[62] T.L. Magnanti, P. Mireault et R.T. Wong. Tailoring benders decomposition for uncapacitated network design. Dans Giorgio Gallo et Claudio Sandi, éditeurs, *Netflow at Pisa*, volume 26 de *Mathematical Programming Studies*, pages 112–154. Springer Berlin Heidelberg, 1986.

[63] A. Migdalas. *Mathematical Programming Techniques for Analysis and : Design of Communication and Transportation Networks*. Linkoping studies in science and technology : Dissertations. Department of Mathematics, Linkoping University, 1988.

[64] M. Minoux. Networks synthesis and optimum network design problems : Models, solution methods and applications. *Networks*, 19:313–360, 1989.

[65] S. M. Oliva et D. Cristian. *Techniques hybrides de propagation de contraintes et de programmation mathématique*. Thèse de doctorat, Université d'Avignon et des Pays de Vaucluse, 2004.

[66] W. B. Powell. A local improvement heuristic for the design of less-than-truckload motor carrier networks. *Transportation Science*, 20:246–257, 1986.

[67] I. Rodriguez-Martin et J. J. Salazar-Gonzalez. A local branching heuristic for the capacitated fixed-charge network design problem. *Computers & Operations Research*, 37:575–581, 2010.

[68] N. Touati, L. Létocart et A. Nagih. Méthodes de décomposition pour l'optimisation discrète. Rapport technique LIPN 2007-1, Laboratoire d'Informatique de Paris-Nord (LIPN), 2007.

[69] D. Tourillon. Méthodes de montée duale pour le problème de conception de réseaux multiproduits avec coûts fixes et capacités. Mémoire de maîtrise, Université de Montréal, 2002.

[70] T.J. Van Roy. A cross decomposition algorithm for capacitated facility location. *Operations Research*, 34:145–163, 1986.

[71] R. T. Wong. Probabilistic analysis of a network design problem heuristic. *Networks*, 15:347–363, 1985.

[72] R.T. Wong. Worst-case analysis of network design problem heuristics. *SIAM Journal on Algebraic Discrete Methods*, 1:51–63, 1980.